# Bachelorarbeit

vorgelegt an der Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt an der

Fakultät Kunststofftechnik und Vermessung

Thema:

## Entwicklung von Methoden zur Erstellung hochpräzise georeferenzierter Karten aus Luftbildern für das autonome Fahren

| | |
|---|---|
| Angefertigt am Institut: | Deutsches Zentrum für Luft- und Raumfahrt<br>Institut für Methodik der Fernerkundung –<br>Photogrammetrie und Bildanalyse (IMF-PBA) |
| Betreuer: | Prof. Dr. Ing. Ansgar Brunn, Dr. Dominik Rosenbaum |
| Prüfer: | Prof. Dr. Ing. Ansgar Brunn, Dr. Dominik Rosenbaum |
| Abgabetermin: | 1. Juni 2020 |

Eingereicht von

Katja Seifert

aus Gerbrunn

Matrikelnummer: 6016061

Würzburg, den 30. Mai 2020

# ERKLÄRUNG ZUR BACHELORARBEIT

Hiermit versichere ich, dass die vorgelegte Bachelorarbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt wurde.

Alle verwendeten Quellen und Hilfsmittel sind angegeben.

Wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Würzburg, den 31.05.2020 Katja Seifert

Unterschrift des/der Studierenden

# Einwilligung zur Überprüfung einer Arbeit mit der Plagiatserkennungssoftware PlagScan

Name: Seifert Vorname: Katja Matr.Nr. 6016061

Adresse: Mühlweg 10a, 97218 Gerbrunn

E-Mail: katja.seifert@student.fhws.de Studiengang: Vermessung und Geoinformatik

Titel der Arbeit: Entwicklung von Methoden zur Erstellung hochpräzise georeferenzierter Karten aus Luftbildern für das autonome Fahren

Betreuer: Prof. Dr. Ing. Ansgar Brunn, Dr.rer.nat. Dominik Rosenbaum

Auf Grund der Zielvereinbarung der FHWS mit dem Bayerischen Staatsministerium für Bildung und Kultus, Wissenschaft und Kunst vom 19. März 2014, Ziffer 2.3, hat die Hochschule entschieden, Studien- und Abschlussarbeiten künftig durch die Plagiatserkennungssoftware PlagScan elektronisch auf Plagiate hin zu überprüfen.

Die zu überprüfenden Arbeiten werden an den Dienst PlagScan übermittelt, dort auf Übereinstimmung mit externen Quellen untersucht und zum Zweck des Abgleichs mit zukünftig zu überprüfenden Studien- und Prüfungsarbeiten gespeichert. **Die befristete Speicherung Ihrer Arbeit in der Datenbank sowie die Weitergabe Ihrer persönlichen Daten im Rahmen der Plagiatsprüfung ist nur mit Ihrer Einwilligung zulässig.**

**Mit einer Unterschrift erkläre ich meine <u>Einwilligung</u>, dass**

- **die von mir vorgelegte und verfasste Arbeit zum Zweck der Überprüfung auf Plagiate hin an PlagScan übermittelt und vorübergehend (5 Jahre) in der von PlagScan geführten Datenbank gespeichert wird;**
- **meine persönlichen Daten (Vorname, Name, studentische E-Mail-Adresse) zusammen mit dem Text digital gespeichert und verwendet werden. Diese Daten sind nur meiner Prüferin oder meinem Prüfer/meinen Prüferinnen oder Prüfern zugänglich.**

**Hinweis:**

*<u>Diese Einwilligungserklärung ist freiwillig.</u> Sie haben die Möglichkeit die Erklärung abzulehnen. Durch die Verweigerung der Einwilligung kann bei Entfernung der persönlichen Angaben und Wahrung der urheberrechtlichen Vorgaben die Plagiatsprüfung nicht verhindert werden. Die Einwilligung zur Speicherung und Verwendung der persönlichen Daten kann jederzeit durch Erklärung gegenüber der Fakultät mit Wirkung für die Zukunft widerrufen werden.*

Ort Würzburg Datum 31.05.2020 Unterschrift Katja Seifert

# Aufgabenstellung

Ziel der Bachelorarbeit ist die Ermittlung der Standardtrajektorien von Fahrzeugen an Kreuzungsbereichen mithilfe von Daten aus Fahrzeugdetektionen.

Die Fahrzeuge werden dabei durch Machine Learning Algorithmen aus mehreren Luftbildern detektiert, welche von einem Hubschrauber aus durch das 4K Kamerasystem des Deutschen Zentrums für Luft- und Raumfahrt (DLR) aufgenommen werden. Um einen aussagekräftigen Datensatz zu erhalten, muss der gewünschte Kreuzungsbereich über längere Zeit beobachtet werden und viele Luftbilder in kurzer Zeit erzeugt werden. Durch eine präzise Georeferenzierung soll die Luftbildorientierung erheblich verbesert werden, was sich direkt auf die Genauigkeit der Detektionsergebnisse auswirkt. Um diese Präzision zu erhalten sollen SAR-GCPs als Bodenreferenzpunkte für eine Bündelblockausgleichung verwendet werden.

Aus den Ergebnissen der Fahrzeugdetektion sollen anschließend die Fahrspuren der Kreuzung ermittelt und durch eine Mittelung der Fahrzeuge die Standardtrajektorien des Kreuzungsbereiches bestimmt werden. Für diesen Schritt soll ein Algorithmus in C++ unter Ubuntu entwickelt werden.

Durch die somit erhaltenen genau bestimmten Trajektorien können die Verläufe der Fahrspuren des Kreuzungsbereiches rekonstruiert werden und somit als Grundlage für präzise Karten dienen.

# Zusammenfassung

Im Zuge dieser Bachelorarbeit wurde eine Methodik entwickelt um aus Fahrzeugdetektionen eines Kreuzungsbereiches Trajektorien der Fahrspuren zu erzeugen. Die Methodik wird in dieser Arbeit vorgestellt.

Dafür wird zunächst der verwendete Datensatz vorgestellt und der Ablauf der Prozessierung der Luftbilder beschrieben. Dieser beinhaltet eine präzise Georeferenzierung mithilfe von sogenannten SAR-GCPs des Deutschen Zentrums für Luft- und Raumfahrt . Die Fahrzeugdetektionen entstehen durch eine Kombination verschiedener Detektionsalgorithmen des Deutschen Zentrums für Luft- und Raumfahrt .

Ein in C++ entwickelter Algorithmus bereinigt den Datensatz und bereitet ihn für die Erkennung von Fahrspuren vor. Der Algorithmus dieser Erkennung teilt die detektierten Fahrzeuge auf die Fahrspuren der Kreuzung auf, indem aus den Detektionsergebnissen ein Graph aufgebaut wird: Dieser entsteht durch die Berechnung aller Vorgänger- und Nachfolgerbeziehungen zwischen Autos. Mithilfe einer Breitensuche werden die Spannbäume des Graphen gefunden, die den Fahrspuren der Kreuzung entsprechen. Mehrspurige Bereiche werden durch Polygone genauer unterteilt. Für jede erkannte Fahrspur können die zugeordneten Autos ausgegeben werden.

Mithilfe der Software MatLab® werden zuletzt die Standardtrajektorien aller Fahrspuren bestimmt, indem eine glättende Spline durch die Autos der Fahrspur gelegt wird. Die somit erhaltenen Trajektorien zeigen in einem Vergleich mit Luftbildern und Kartengrundlagen gute Ergebnisse.

# Inhaltsverzeichnis

# 1 Einführung

Im Folgenden wird auf die Motivation dieser Bachelorarbeit eingegangen und die Zielsetzung genauer beschrieben.

## 1.1 Motivation

Das Projekt D.MoVe (Datengetriebenes Mobilitäts- und Verkehrsmanagement) des Deutschen Zentrums für Luft- und Raumfahrt umfasst verschiedene Forschungsbereiche mit dem Ziel Ansätze für das Verkehrsmanagement der Zukunft zu entwickeln. Dafür sind präzise Kartengrundlagen ein wichtiger Aspekt der Steuerung von Verkehrsflüssen - besonders mit der Beteiligung von autonomen Fahrzeugen.

In den Bereichen der Fahrzeugnavigation und des autonomen Fahrens sind präzise Kartierungen von Fahrspuren wertvoll. Denn vor allem in komplizierten urbanen Kreuzungsbereichen ist es wichtig, die einzelnen Fahr- und Abbiegespuren unterscheiden zu können. Bisherige Karten haben für diese Anwendung nicht genug Daten: Straßen werden häufig über ihre Mittelachse definiert anstatt die einzelnen Fahrspuren darzustellen und Kreuzungen werden lediglich durch das Kreuzen der Straßenachsen repräsentiert. Kartengrundlagen für das autonome Fahren hingegen benötigen präzise Informationen über die Fahrspuren um das Fahrzeug sicher durch den Straßenverkehr manövrieren zu können. Dazu können sogenannte HD Maps verwendet werden, welche die benötigten Informationen für autonome Fahrzeuge enthalten. Darin können Informationen von Straßenverläufen, Straßenmarkierungen bis zu den Positionen

und Bedeutungen von Straßenschildern dokumentiert sein.

Die Erstellung der bisherigen HD Maps wird meist mit einem Messfahrzeug realisiert: Ein Auto, das mit ausreichenden Sensoren und Messsystemen (Laserscanner, LiDAR, GPS, Kameras) ausgestattet wird, fährt die zu kartierenden Bereiche ab. Durch eine ausführliche Postprozessierung entstehen hochgenaue Karten bzw. 3D Modelle, welche die reale Situation bestmöglich abbilden. Somit können nicht nur die Fahrspuren, sondern auch Verkehrszeichen, Fahrbahnmarkierungen und -begrenzungen usw. aufgezeichnet werden. (Vardhan 2017)
Da solche Karten allerdings darauf angewiesen sind, alle gewünschten Bereiche mit einem Messfahrzeug zu befahren, lohnt sich dieser Ansatz nicht für die Erstellung großflächiger Kartengrundlagen. Diese Arbeit verwendet als Datengrundlage daher Luftaufnahmen, welche schneller größere Bereiche erfassen können.

## 1.2 Zielsetzung

Das Ziel dieser Arbeit soll es sein, aus Luftbildern möglichst präzise Fahrspurtrajektorien für Kreuzungsbereiche zu erzeugen. Diese könnten HD Maps unterstützen oder bisherige Kartengrundlagen erweitern.

In dieser Arbeit soll ein Algorithmus entwickelt werden, welcher aus den Ergebnissen der Verkehrserfassung des DLR gemittelte Trajektorien von Fahrzeugen in Kreuzungsbereichen ermittelt. Dabei soll auch ein Verfahren zur präzisen Georeferenzierung von Luftbildern angewendet werden, um die Genauigkeit des Datensatzes zu erhöhen. Das erhaltene Ergebnis soll für eine weitere Verwendung aufbereitet und mit aktuellen Kartengrundlagen verglichen werden.

# 2 Datengrundlage

Im folgenden Kapitel sollen sowohl die Herkunft als auch der Aufbau der verwendeten Daten vorgestellt werden. Auch auf das Preprocessing der Daten wird eingegangen und dieses gegebenenfalls näher erläutert. Außerdem werden die Stärken und Schwächen, sowie mögliche Probleme der Datensätze dargestellt.

## 2.1 Kamerasystem und Luftbilder

Der in dieser Arbeit verwendete Datensatz von Luftbildern wurde mit einem Kamerasystem des Deutschen Zentrums für Luft- und Raumfahrt aufgenommen. Dieses Kamerasystem und der damit erzeugte Luftbilddatensatz werden im Folgenden genauer beschrieben.

### Das Kamerasystem des DLR

Das 4K Kamerasystem wurde von der Abteilung Photogrammetrie und Bildanalyse des Instituts für Methodik der Fernerkundung entwickelt. Es beinhaltet drei handelsübliche Kameras, wobei zwei der Kameras seitlich zur Flugrichtung ausgerichtet sind und die dritte Kamera nach Nadir ausgerichtet ist. Zusätzlich kann die Nadir Kamera Aufnahmen im nahen Infrarotbereich sowie Videos in 4K-Auflösung erzeugen. Weitere Komponenten sind ein GNSS Empfänger, eine IMU, drei Recheneinheiten sowie ein System zur Real-Time Datenübertrageung an eine Bodenstation via C-Band Da-

tenlink. Dieses System wurde so konzipiert, dass es an einen Hubschrauber montiert und von diesem aus betrieben werden kann. (Kurz, Rosenbaum u. a. 2014)

Abbildung 2.1: 4K Kamerasystem an Hubschrauber BO105

## Testdatensatz eines Kreuzungsbereichs in Senden (Ulm)

Für diese Arbeit wurde ein Datensatz an Testbildern verwendet, die am 27.09.2018 über einer Kreuzung in Senden (bei Ulm) aufgenommen wurde. Dafür wurde das 4K Kamerasystem an einem Hubschrauber des Typs BO 105 montiert. Es wurden Bilder der linken und rechten Kameras mit einer Frequenz von 1Hz aufgenommen. In dem Aufnahmenzeitraum von insgesamt ca. 10min - aufgeteilt auf eine halbe Stunde - wurden 633 Bilder der ausgewählten Kreuzung erzeugt. Abbildung 2.2 zeigt den Kreuzungsbereich aus einem der Luftbilder. Während des Fluges wurde die Flughöhe von ca. 300m auf ca. 550m erhöht, sodass in etwa die Hälfte der Bilder eine geringere geometrische Auflösung besitzen und einen größeren Bereich der Kreuzung zeigen.

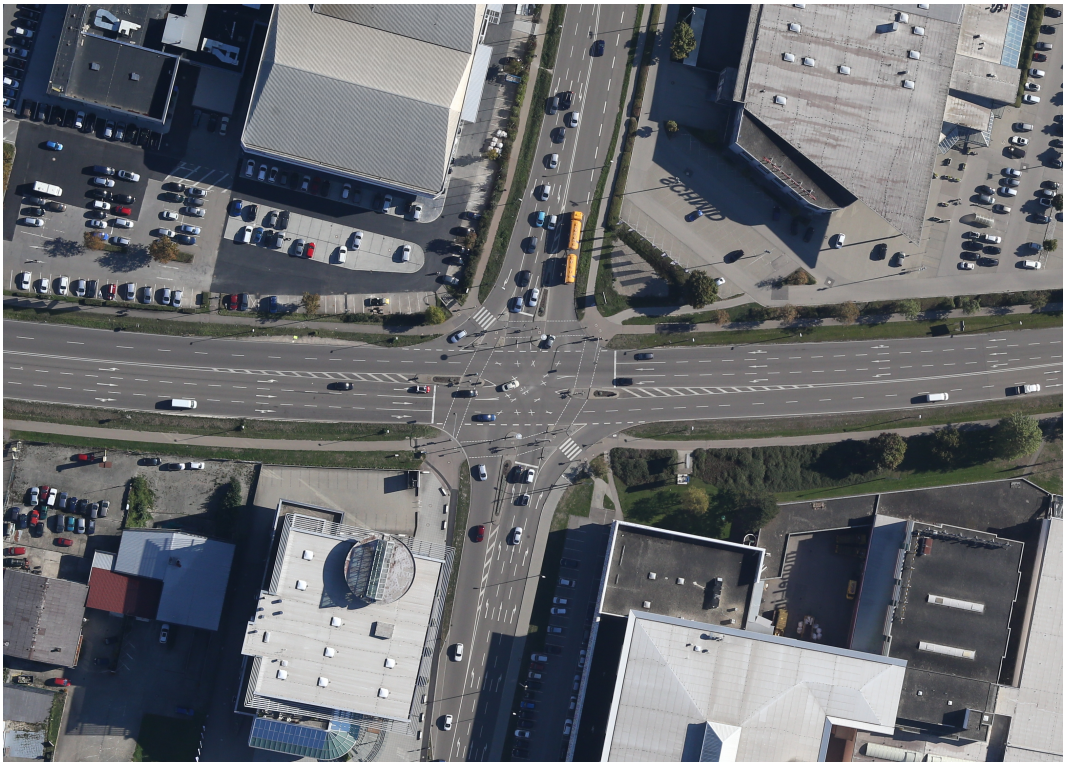


Abbildung 2.2: Kreuzungsbereich des Datensatzes aus Senden

## 2.2 Georeferenzierung der Luftbilder

Bevor die Koordinaten und Ausrichtungen der Autos aus den Luftbildern bestimmen werden können, müssen diese georeferenziert werden. Dies geschieht mittels einer Bündelblockausgleichung der vororientierten Bilder - wobei die Vororientierung durch das 4K Kamerasystem (GNSS und IMU) gegeben ist. Um durch die Bündelblockausgleichung jedoch auch einen genauen Bezug zum übergeordneten Koordinatensystem (UTM) herstellen zu können sind genaue Bodenreferenzpunkte - auch „Ground Control Points“ (GCPs) genannt - nötig. Die Punkte sollten gut über das Gebiet verteilt sein und möglichst vielen Bildern zugeordnet werden können. Um das zu realisieren bieten sich die SAR GCPs des Deutschen Zentrums für Luft- und Raumfahrt an.

## Herkunft der SAR GCPs

Das Verfahren zur Bestimmung der SAR GCPs wurde im Rahmen des Projekts DriveMark® entwickelt. (Jung 2018)

Wie der Name bereits nahelegt, werden die SAR GCPs aus den Aufnahmen eines „Synthetic Aperture Radar“ erzeugt. Das DLR verwendet dafür die Messungen des TerraSAR-X Satelliten. Dabei wird das stark reflektierende Verhalten von senkrecht stehenden und zur Messung ausgerichteten Objekten genutzt, um die Fußpunkte von Laternenmasten, Pfosten und Verkehrsschildern koordinatenmäßig zu bestimmen. Da solche Objekte für das Radar wie Retroreflektoren fungieren, sind sie in der Auswertung in Form von hellen Punkten deutlich identifizierbar. Ihre Koordinaten werden aus der Mittelbildung mehrerer Messungen (mind. 3) gebildet, wobei an die Messungen zunächst Korrekturen von ionosphärischen und atmosphärischen Effekten sowie von Einflüssen der Plattentektonik angebracht werden.

Die Genauigkeit der Ergebnisse hängt von mehreren Faktoren ab. Dazu zählen, neben der Anzahl der Messungen und der Genauigkeit des SAR selbst, auch die Beschaffenheit des Objektes und seine nähere Umgebung. Letztere sollte idealerweise einer ebenen Fläche entsprechen. Abweichungen davon werden durch das sog. „Signal to Clutter Ratio “ berücksichtigt.

Die somit erhaltenen Punkte werden nach ihrer Genauigkeit gefiltert, sodass lediglich Punkte mit einer Genauigkeit von unter 10cm als SAR GCPs gelten. Auf diese Weise ist es möglich ohne eine direkte Messung vor Ort präzise Bodenreferenzpunkte mit Genauigkeiten im Bereich von wenigen Zentimetern zu erzeugen.
(Kurz, Krauß u. a. 2019)

## Ablauf der Georeferenzierung

Um die Luftbilder des Testdatensatzes vollständig zu orientieren werden die SAR GCPs als Bodenreferenzpunkte verwendet. Anschließend erfolgt eine Bündelblockausgleichung. Der Ablauf dieser Schritte wird im Folgenden genauer erläutert.

### Einmessung der Bodenreferenzpunkte in den Bildern

Für die Einmessung der Punkte wurden die vorhandenen SAR GCPs innerhalb des von den Luftbildern abgedeckten Bereichs bestimmt und in mehreren Bildern identifiziert. Für jeden Punkt wurden die Bildkoordinaten aus mehreren Luftbildern möglichst genau erfasst und in einer gemeinsamen Datei gesammelt.
Bei der manuellen Erfassung der SAR GCPs muss darauf geachtet werden, dass der Fußpunkt der Objekte möglichst genau bestimmt wird. Dies konnte vor allem in abgedunkelten Bereichen oder bei fehlendem Kontrast zwischen dem Objekt und dem Untergrund zu Problemen führen. Auch die senkrechte Sicht in den Bildzentren kann dazu führen, dass die Fußpunkte der Objekte nicht genau zu erkennen sind. Die genaue Differenzierung von zwei nahe beieinander liegenden Punkten konnte ebenfalls zu Schwierigkeiten führen. Aufgrund dieser Einschränkungen konnten nicht alle SAR GCPs als Bodenreferenzpunkte einbezogen werden.

### Bündelblockausgleichung

Um die Bündelblockausgleichung berechnen zu können, müssen alle vororientierten Luftbilder, die UTM-Koordinaten sowie die Bildkoordinaten der SAR GCPs vorliegen. Anschließend werden in den Luftbildern zunächst mögliche Verknüpfungspunkte für die relative Orientierung gesucht. Diese Merkmalsextraktion geschieht mithilfe des

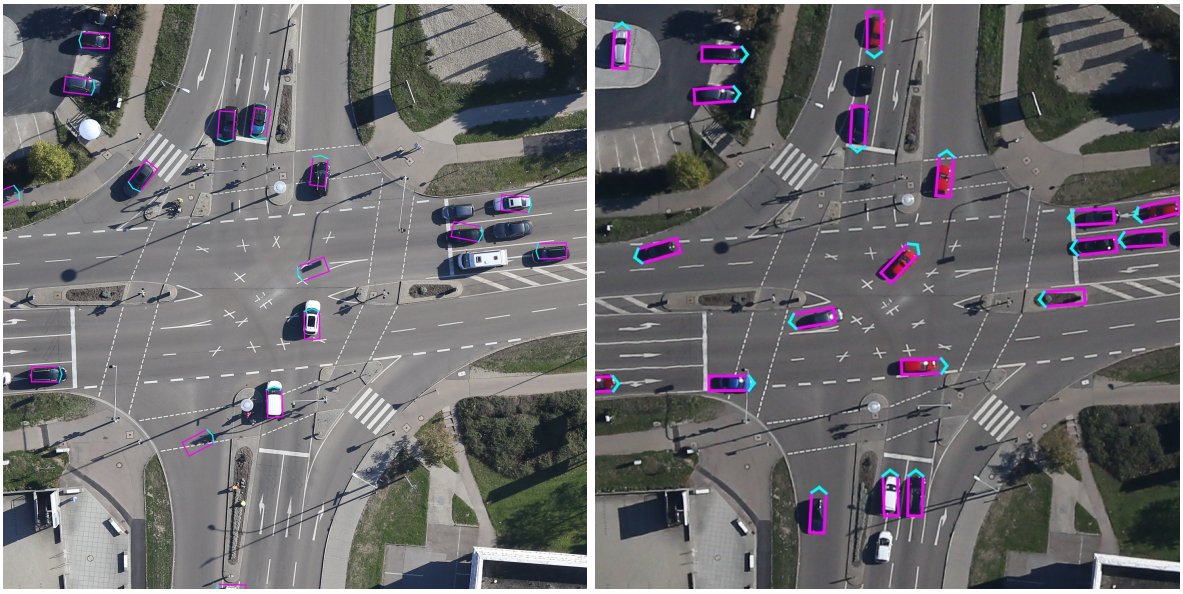
sog. SIFT Algorithmus. Dadurch können die Luftbilder über gemeinsame Verknüpfungspunkte einander zugeordnet werden. Mithilfe der Bodenreferenzpunkte kann nun eine Bündelblockausgleichung berechnet werden, welche auch übergeordnete Koordinaten für die Bilder ermittelt. Somit liegen nun präzise georeferenzierte Luftbilder des Kreuzungsbereiches vor. Auf dieser Grundlage ist es möglich Autos, welche in einzelnen Luftbildern erkannt werden, mit Koordinaten zu verknüpfen und so die Detektionen aus mehreren Bildern in Bezug zu setzen. Abschließend wird die Georeferenzierung ausgegeben und Orthobilder erstellt, welche als Grundlage für die Fahrzeugdetektion verwendet werden.

## 2.3 Detektion der Autos

In den Orthobildern können über verschiedene Methoden Fahrzeuge automatisch detektiert werden. Hier werden zwei Machine Learning Algorithmen, die im Deutsche Zentrum für Luft- und Raumfahrt bereits bestehen, angewendet und ihre Ergebnisse kombiniert.

### 2.3.1 Erkennung durch einen Deep Learning Algorithmus

Ein neueres Projekt des Deutschen Zentrums für Luft- und Raumfahrt ist der Deep Learning Algorithmus aus (Azimi, Vig u. a. 2018), welcher ein Convolutional Neural Network ist. Dieser wurde für Luftbilder mit einer Bodenauflösung von 13cm trainiert, weshalb die Ergebnisse für die hier verwendeten Orthobilder (mit Auflösungen von ca. 5cm und 7cm) nicht optimal, aber dennoch gut sind.



(a) Detektionen bei 5cm Auflösung (b) Detektionen bei 7cm Auflösung

Abbildung 2.3: Detektionen des Deep Learning Algorithmus

Der Deep Learning Algorithmus bearbeitet jedes Bild einzeln und gibt seine Detektionsergebnisse in einer XML-Datei und visualisiert in einer Bilddatei aus. In der XML-Datei ist für jedes Auto die Bounding Box mit Zentrum in Bildkoordinaten, die Ausrichtung des Autos und weitere Daten, wie die Sicherheit der Detektion, angegeben.

Abbildung 2.3 zeigt die Detektionsergebnisse zweier Luftbilder - zugeschnitten auf den Kreuzungsbereich. Insgesamt wurden dort viele der sichtbaren Autos erkannt - allerdings kommt es auch zu Fehldetektionen: Bei einer Auflösung von 5cm wurden auch manche Schatten von Straßenlaternen oder Ampeln als Fahrzeuge erkannt. Diese Fehler traten bei 7cm Auflösung nicht mehr auf - dafür wurden aber gelegentlich die Verkehrsinseln detektiert. Ein Ansatz diese Fehldetektionen herauszufiltern wird in Kapitel 4.2.3 vorgestellt. Die erkannten Ausrichtungen der Autos sind in diesen Daten oft nicht so präzise und gelegentlich auch um 180° verdreht. Durch die

Kombination der beiden Detektionsansätze in 2.3.3 können die Ausrichtungen jedoch noch verbessert werden.

Da der Ort des detektierten Fahrzeuges nur in Bildkoordinaten vorliegt und sich auch die Ausrichtung auf die Achsen des Bildes bezieht, muss die Georeferenzierung nachträglich eingebaut werden. Das dazu verwendete Skript nutzt Informationen aus einem Geländemodell - hier wurde ein hochaufgelöstes DSM der Vermessungsverwaltung gewählt. Mithilfe eines weiteren Skripts werden die Daten in eine car_detected-Datei umgewandelt. Deren Aufbau und Inhalt wird in einem späteren Abschnitt erläutert.

### 2.3.2 Erkennung durch einen Mustererkennungsalgorithmus

Eine weitere Methode der Fahrzeugdetektion aus Luftbildern sind Mustererkennungsalgorithmen der Bildverarbeitung. Hier wurde im Deutschen Zentrum für Luft- und Raumfahrt bereits ein Machine Learning Algorithmus entwickelt, welcher Support-Vektor-Maschinen (SVM) und Adaptive Boosting für die Klassifizierung von Fahrzeugen verwendet. (Leitloff u. a. 2014). Dieser Algorithmus dient als Grundlage für die Verkehrserfassung des DLR. Durch die Klassifizierung werden u.a. die Koordinaten des Mittelpunktes (sowohl in Bild- als auch in UTM-Koordinaten) und die Ausrichtung des Fahrzeuges bestimmt. In Abbildung 2.4 ist ein Ausschnitt der Detektionsergebnisse zu sehen. Insgesamt wurden die Koordinaten und Ausrichtungen in dieser Methode genauer erkannt als durch den Deep Learning Algorithmus - dafür wurden allerdings weniger Fahrzeuge detektiert.

Alle Detektionen eines Bildes werden in einer car_detected-Datei ausgegeben. Neben den Detektionen werden auch sogenannte navteq_roads-Dateien erzeugt, welche Informationen über die Straßen innerhalb des Bildes enthalten.

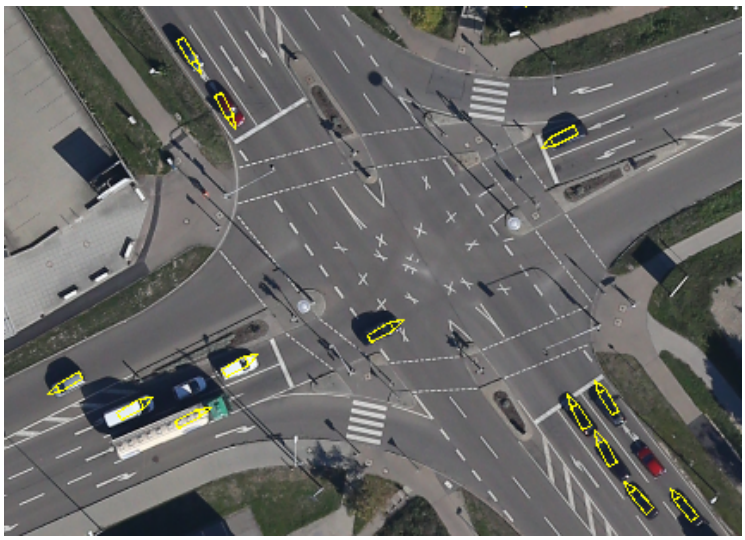


Abbildung 2.4: Fahrzeugsetektionen der Mustererkennung

### 2.3.3 Kombination der Detektionen

Da durch die erste Methode im Vergleich zur zweiten Methode zwar viele Autos erkannt wurden, deren Ausrichtung jedoch nicht so genau erfasst werden konnten, wurde entschieden die Ergebnisse beider Methoden zu kombinieren. Ziel war es, identische Detektionen der beiden Ergebnisse zu finden und sie zu einer einzigen verbesserten Detektion zu machen. Dafür war es wichtig, beide Methoden auf denselben Orthobildern anzuwenden.

Für die beiden Detektionsergebnisse eines Bildes werden Paare von Detektionen gefunden, deren Mittelpunkte im Rahmen einer Toleranz übereinstimmen. Ist ein solches Paar erkannt werden alle Werte der Deep Learning Detektion übernommen - mit Ausnahme der Ausrichtung, welche aus dem Ergebnis der Mustererkennung verwendet wird. Zuletzt werden alle somit verbesserten Detektionen als auch alle Detektionen des Deep Learning Algorithmus, die nicht Teil eines Paares sind, in eine neue car_detected-Datei gespeichert. Diese Daten werden in den folgenden Kapiteln verwendet und sind damit die Grundlage dieser Arbeit.

### 2.3.4 Ausgabedateien

Durch die Detektion der Autos entstehen zwei wichtige Dateien:

**car_detected-Datei:** In einer car_detected-Datei befinden sich alle Detektionen eines Luftbildes. Dabei stellt jede Zeile ein Detektionsergebnis mit den Attributen aus Tabelle 2.1 dar. Eine genauere Beschreibung der Attribute ist in (Seifert 2019) zu finden. Für die hier entwickelten Algorithmen sind jedoch nur die UTM-Koordinaten und die Ausrichtung der Fahrzeuge relevant.

| ID | Bildkoordinaten | | UTM–Koordinaten | | Geschwindigkeits-begrenzung der Straße | Ausrichtung | | Qualität der Detektion | Fahr-zeug-klasse |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 3288.50 | 155.00 | 577029.734... | 5353421.871... | 100 | 0.621382 | -0.783508 | 0.99 | 1 |
| 1 | 4959.25 | 283.25 | 577062.097... | 5353352.515... | 100 | -0.640120 | 0.768275 | 0.99 | 1 |
| ... | ... | | ... | | ... | ... | | ... | ... |

Tabelle 2.1: Attribute und Inhalt einer car_detected-Datei

**navteq_roads-Datei:** Diese Datei beinhaltet Informationen über den Straßenverlauf. Die Straßendaten werden aus OpenStreetMap Daten erzeugt. Dabei werden Straßen durch Stützpunkte („Nodes“) dargestellt - der Straßenverlauf wird durch die Verbindungen zwischen den entsprechenden Nodes rekonstruiert. Zusammengehörige Straßenabschnitte werden mit einer eindeutigen Nummer und Navteq-ID gekennzeichnet. Außerdem besitzt ein Straßenabschnitt auch eine Straßenkategorie, eine Kategoriesierung der bekannten Fahrstreifen sowie eine Geschwindigkeitsbegrenzung.

Eine navteq_roads-Datei eines Luftbildes enthält - wie in Tabelle 2.2 dargestellt - in jeder Zeile eine Node. Die Straßenkategorie beschreibt verschiedene Nutzungsarten der Straßen, beispielsweise Fußgängerwege, Wohnstraßen oder Autobahnen.

| Nummer des Abschnittes | Navteq-ID des Abschnittes | Reihenfolge der Node | Straßen-kategorie | Geschwindigkeits-begrenzung | Fahr-streifen-kategorie | UTM–Koordinaten |
|---|---|---|---|---|---|---|
| 0 | 244481226 | 0 | 22 | 200 | 1 | 576927.13 5353474.50 |
| 0 | 244481226 | 1 | 22 | 200 | 1 | 576935.88 5353472.00 |
| ... | ... | ... | ... | ... | ... | ... |

Tabelle 2.2: Attribute und Inhalt einer navteq_roads-Datei

Im Falle der Kreuzung in Senden gehören die relevanten Straßen der Kategorie 9 an, was der OpenStreetMap-Kategorie „highway: secondary“ (OpenStreetMap-Wiki 2019) entspricht.

Da die hier verwendeten Luftbilder größtenteils denselben Bereich abdecken, enthalten die navteq_roads-Dateien der Bilder viele Straßenstützpunkte mehrfach.

## 2.4 Tracking der Autos

Die Verkehrserfassung des Deutschen Zentrums für Luft- und Raumfahrt ermöglicht nicht nur die Detektion, sondern auch das Tracking von bereits detektierten Fahrzeugen. Dabei wird versucht jedes Auto eines Bildes in einem oder mehrerer zeitlich folgenden Bilder wieder zu finden, wodurch Rückschlüsse auf die Fahrtrichtung und Geschwindigkeit gezogen werden können. Die dafür verwendeten Bilder werden in einem sogenannten Burst zusammengefasst. (Leitloff u. a. 2014)

Aufgrund der größeren Anzahl an erfolgreichen Tracking-Ergebnissen wird in diesem Datensatz das Tracking mit zwei Bildern gewählt.

### 2.4.1 Ablauf des Trackings

Für das Tracking müssen beide Orthobilder des Bursts sowie die car_detected- und navteq_roads-Datei des ersten Bildes angegeben sein. Der Algorithmus der Verkehrserfassung versucht durch Template-Matching, die Autos des ersten Bildes im zweiten Bild wiederzufinden (Leitloff u. a. 2014). Ist eine solche Zuordnung möglich, wird die Detektion im zweiten Bild einer neuen car_detected-Datei hinzugefügt und die Information des Trackings in der cpdb-Datei gespeichert.

### 2.4.2 Ausgabedateien

**car_detected Datei:** Es werden in jedem Burst zwei car_detected-Dateien ausgegeben: Eine Kopie der Eingangsdatei des ersten Bildes und eine neue Datei für das zweite Bild. Letztere beinhaltet alle Autos, die in diesem Bild detektiert und als übereinstimmend mit einem Auto des ersten Bildes erkannt wurden. Da eine solche eindeutige Zuordnung recht schwierig ist, gelingt dies nie bei allen Detektionen. Es bietet sich daher an die Detektionsergebnisse des Trackings zusätzlich zu den direkten Detektionen der Autos als Datengrundlage zu verwenden.

**cpdb-Datei:** Diese Datei fasst das Ergebnis des Trackings zusammen: Neben Informationen über die Anzahl der getrackten Objekte und die beiden verwendeten Luftbilder sind auch die einzelnen Zuordnungen von Detektionen gespeichert. Für jedes erfolgreiche Tracking sind die Positionen der beiden Detektionen in Bild- und UTM-Koordinaten angegeben. Um weitere Informationen, wie bspw. die Ausrichtungen der beiden Detektionen, zu erhalten, müssen diese über ihre Koordinaten in der entsprechenden car_detected-Datei gefunden werden.

Die in diesem Kapitel beschriebenen Daten dienen als Grundlage für den Algorithmus der Fahrspurerkennung. Es werden die Ausgabedateien der Detektion und des Trackings sowie die erzeugten Orthobilder verwendet, um Fahrspuren automatisch zu erkennen und ihre Standardtrajektorien zu bestimmen.

# 3 Visualisierung der Daten

Die Daten der Verkehrserfassung in Senden sowie alle Ergebnisse des Algorithmus, die in dieser Arbeit entstehen, liegen lediglich als ASCII-Dateien vor - mit Ausnahme der Orthobilder. Eine Kontrolle von Zwischenergebnissen oder eine Analyse dieser räumlichen Daten sind damit deutlich erschwert. Eine einfache Lösung dafür bieten Geoinformationssysteme wie ArcMap und QGIS. Da das Programm QGIS (QGIS Development Team 2020) unter der „GNU General Public License“ steht und damit leichter zugänglich ist, wurden die Daten in dieser Arbeit in QGIS visualisiert. Dabei wurde die Version 2.18.17 unter Ubuntu 18.04.4 LTS verwendet.

Um die Daten und Ergebnisse dieser Arbeit visuell noch besser einordnen zu können, bietet sich die Verwendung eines der Orthobilder an. Das ausgewählte Orthobild (vgl. Abbildung 3.1) bildet den relevanten Bereich der Kreuzung großzügig ab und eignet sich damit sehr gut als Hintergrund der Daten.

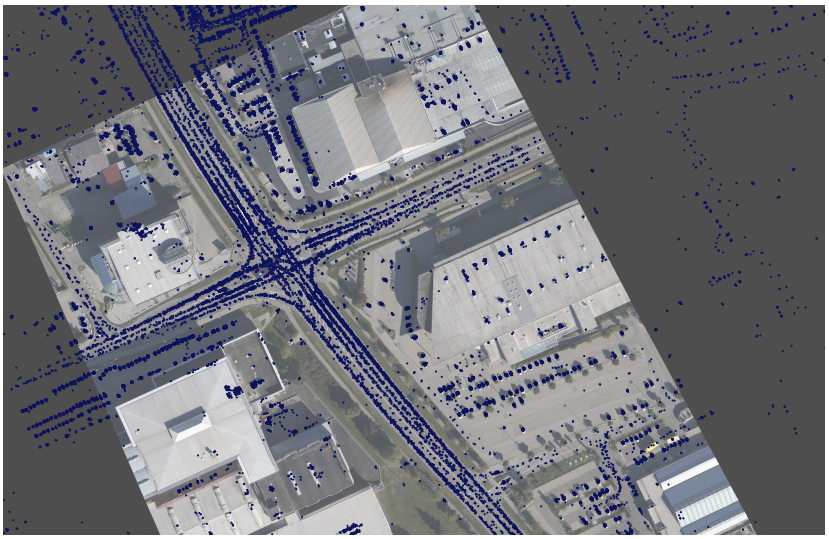


Abbildung 3.1: Orthobild mit allen detektierten Fahrzeugen

# 4 Algorithmus

Um aus den Datengrundlagen Standardtrajektorien von Fahrspuren zu erhalten, wurde zunächst ein C++ Programm entwickelt, welches die Daten einliest, bereinigt und Fahrspuren erkennt. Dieses Programm gibt zuletzt alle Autos einer Fahrspur aus und ermöglicht es dadurch eine gemittelte Trajektorie durch die Daten zu finden. Der gesamte Ablauf wird in den folgenden Kapiteln anhand des vorhandenen Datensatzes erläutert.
Der Algorithmus wurde in der C++17 Version implementiert. Dafür wurde die Software Visual Studio Code (Microsoft 2020) in der Version 1.45.1 verwendet.

## 4.1 Einlesen der Daten

### 4.1.1 Grundsätzliche Funktionen

Da das Einlesen der verschiedenen Textdateien auf demselben Ablauf basiert, wird dieser in den Funktionen der „import_tools.cpp“ Datei allgemein implementiert.

Um eine Textdatei damit einzulesen muss der Pfad des übergeordneten Ordners sowie ein Regex-Pattern angegeben werden, um die gesuchten Dateien anhand ihrer Namen zu finden. Diese Dateien müssen nicht direkt innerhalb des Ordners liegen sondern können auch in Unterordnern liegen, da das Programm rekursiv alle Ordner und Unterordner nach übereinstimmenden Dateinamen durchsucht. Das Regex-Pattern dient dazu, die entsprechenden Dateien zu finden. Es wurde eingebaut da

es vorkommen konnte, dass die Dateinamen alleine nicht ausreichen. Beispielsweise gab es Dateien, die als Namen „_cpdb1_OL0344“ hatten, wobei der letzte Teil das zugehörige Luftbild darstellt und daher bei jeder Datei anders ist. Da aber alle Dateien, die mit „_cpdb1_“ beginnen gesucht waren, ließ sich das über das Regex-Pattern `\^\_cpdb1\_.*` darstellen.

Da die Informationen in den einzulesenden Dateien auf ähnliche Art (zeilenweise) gespeichert sind, können sie ebenfalls durch eine gemeinsame Funktion eingelesen werden. Dabei wird jede Zeile einer Datei als `string` gespeichert und über eine Trennung bei Leerzeichen als `vector<string>` aller Daten dieser Zeile verfügbar gemacht.

### 4.1.2 Import der detektierten Autos

Der Ablauf sowie die grundsätzliche Struktur des Imports orientiert sich an der Arbeit (Seifert 2019).

#### Aufbau der Klasse Car

Um die detektierten Autos gut in den Algorithmus einbeziehen zu können, wurde eine Klasse für diese erzeugt. Alle Attribute und Methoden sind im Klassendiagramm 4.1 aufgelistet.
Die Attribute `img_coordinates`, `utm_coordinates` und `direction` werden aus den Detektionen übernommen. Um die Beziehungen der Objekte untereinander und ihren Bezug zu den Straßenabschnitten zu berücksichtigen gibt es zusätzlich die folgenden Attribute:

**navteq_ID:** Die Navteq-ID des Straßenabschnitts, dem das Auto am nächsten ist.

**street_dist:** Die Distanz zu diesem Straßenabschnitt (vgl. Kapitel 4.2.2)

**is_tracked:** Speichert den Tracking-Status eines Autos. Wenn für dieses Auto im Vorgänger- oder Nachfolgebild ein Pendant existiert, gilt es als getrackt und wird mit `is_tracked = True` vermerkt. (vgl. Kapitel 4.1.3)

**predecessors/ successors:** Ein `vector<Car*>` mit `pointern` zu den Vorgänger- bzw. Nachfolger-Objekten (vgl. Kapitel 4.3.2)

Da die oben genannten Attribute erst im weiteren Verlauf des Algorithmus relevant werden, sind genauere Erklärungen zu diesen in den späteren Kapiteln zu finden. Die Klasse Car besteht aus einer header Datei („class_Car.hpp“), welche die Deklaration der Klasse und ihrer Methoden sowie die (inline-)Definitionen von `get-` und `set`-Methoden enthält.

In der zugehörigen Code-Datei („class_Car.cpp“) sind die Definitionen der Konstruktoren, der Klassenvariablen `count` sowie der weiteren Methoden enthalten. Neben der in (Seifert 2019) vorgestellten Methode `convertCarToString()`, mit deren Hilfe eine leerzeichgetrennte `string` Darstellung eines Car-Objekts erstellt wird, wurden der Klasse folgende Aspekte hinzugefügt: Die Methoden `add_predecessors()` und `add_successors()` ermöglichen es, einen oder mehrere Vorgänger/ Nachfolger (welche ebenfalls Car-Objekte sind) zu hinterlegen. Dabei werden die `pointer` zu diesen Car-Objekten dem entsprechenden Attribut (`predecessors`/ `successors`) hinzugefügt und das Attribut `is_tracked` auf `True` gesetzt, da offenbar eine Vorgänger-Nachfolger-Beziehung für dieses Objekt existiert.

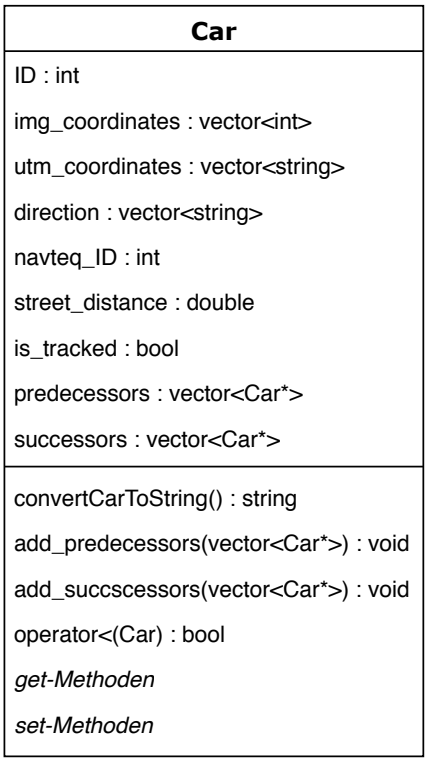



Abbildung 4.1: Klassendiagramm der Klasse Car

Außerdem ist es innerhalb eines Algorithmus nicht immer nötig oder möglich bei der Instanziierung eines Objektes Werte für alle Attribute mitzugeben. Deshalb ist es hilfreich verschiedene Konstruktoren mit unterschiedlichen Parameterlisten zur Verfügung zu stellen. Einer davon ist der Standardkonstruktor, welcher keine Parameter entgegennimmt und damit ein „leeres“ Auto erzeugt. Ein solches besitzt bspw. die `ID` „-1“.

Ein weiterer Aspekt ist der < Operator: Der Operator < kann zwei Objekte vergleichen und liefert `True` zurück, falls das erste Objekt kleiner als das zweite ist. Andernfalls wird der Wert `False` zurückgegeben. Dieser Operator wird daher häufig für Sortierungsalgorithmen verwendet, wie beispielsweise die Funktion `std::sort()` der C++ Standardbibliothek (*CPP-Reference: std::sort* 2020). Da es für einen Algorithmus bei Objekten einer eigenen Klasse nicht ersichtlich ist, wann ein Objekt als „kleiner“ zu

verstehen ist, muss dieser Operator für Objekte der Klasse überladen werden - sofern der < Operator im Programmcode mit solchen Objekten aufgerufen wird. Weil eine Sortierung nach den Koordinaten der Car-Objekte in diesem Projekt sinnvoll ist, wird der Operator < für die Klasse Car wie folgt definiert:
Zunächst werden die $UTM_X$ Werte verglichen und entscheiden, ob das Objekt $a$ kleiner als Objekt $b$ ist. Sind diese Werte gleich, entscheidet die $UTM_Y$ Komponente:

$$\begin{array}{llllll} a < b & wenn: & a.UTM_X < b.UTM_X & & & \\ oder & wenn: & a.UTM_X = b.UTM_X & und & a.UTM_Y < b.UTM_Y \end{array}$$

Dies kann, wie im nächsten Unterkapitel erklärt, auch verwendet werden um mehrfach in einem `vector` vorkommende Objekte zu löschen.

### Einlesen der Car Objekte

Für das Einlesen der Daten aus den car_detected-Dateien sind die Funktionen der Datei „import_car_data.cpp “zuständig. Dabei werden, mithilfe den Funktionen aus Kapitel 4.1.1, alle car_detected-Dateien gefunden und deren Zeilen eingelesen. Aus den Daten einer Zeile wird automatisch ein Car-Objekt erzeugt und zum Schluss ein `vector<Car>` mit allen eingelesenen Autos zurückgegeben.
Außerdem ermöglicht es die Funktion `writeCarsInFile()` einen vector mit Car-Objekten als CSV-Datei auszugeben.

### 4.1.3 Import der getrackten Autos

Wie in Kapitel 2.4.2 erläutert ist es sinnvoll, beide Ausgabedateien des Tracking Algorithmus einzulesen.

#### car_detected-Dateien

Die durch das Tracking entstandenen car_detected-Dateien werden wie in Kapitel 4.1.2 eingelesen. Da mindestens die Hälfte der darin enthaltenen Detektionen bereits in den Daten des vorherigen Kapitels vorhanden sind, kommt es beim Einlesen der neuen Dateien zu doppelt vorkommenden Autos. Um dieses Problem zu beheben werden diese Duplikate durch das Nutzen eines `sets` entfernt, bevor der `vector<Car>` zurückgegeben wird.

Ein `Set` stellt in C++ eine sortierte Liste dar, die keine doppelten Elemente enthalten kann. Dabei werden die Elemente intern über den < Operator nach ihrer „Größe“ sortiert. Um Duplikate zu erkennen wird der Größenvergleich über den Operator in beiden Richtungen geprüft: Wenn weder $a < b$ noch $b < a$ gilt, müssen die Elemente $a$ und $b$ gleich sein.(*CPP-Reference: std::set* 2020) Nachdem somit alle Dateien eingelesen wurden, wird das `Set` zu einem `vector` umgewandelt und dieser zurückgegeben.

Dass durch die zusätzlichen car_detected-Dateien neue Informationen gewonnen wurden, ist in Abbildung 4.2 gut zu erkennen: Die Ergebnisse der kombinierten Detektionen (vgl. Kapitel 2.3.3) sind in blau, die zusätzlichen Detektionsergebnisse durch das Tracking in rot dargestellt.

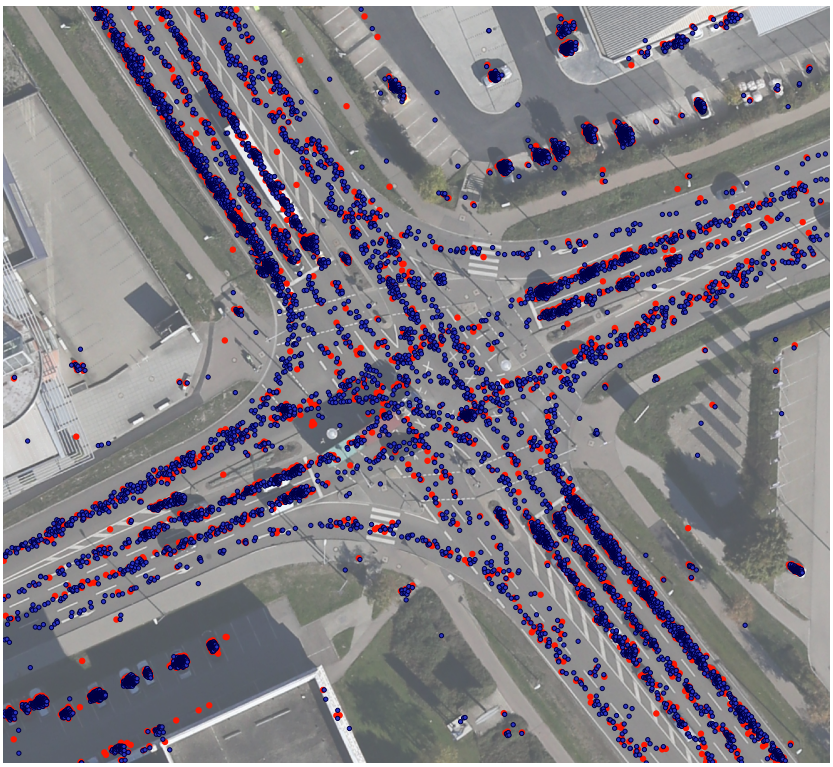


Abbildung 4.2: Detektionen aus dem Tracking Algorithmus

### _cpdb-Dateien

Die Tracking-Informationen eines Autos aus den cpdb-Dateien beschränken sich auf die Koordinaten des getrackten Autos an der Anfangs- und Endposition. Um die beiden zugehörigen Car-Objekte zu finden, muss der bisherige `vector` mit den Autos der car_detected-Dateien nach den Koordinaten durchsucht werden. Dabei musste folgendes Problem berücksichtigt werden:
Die in den Dateien gespeicherten Koordinatenwerte waren auf unterschiedlich viele Nachkommastellen gerundet. Dadurch ist nicht nur ein einfacher Vergleich der Werte als `strings` sondern auch als `double`-Zahlenwerte ungültig. Um diese Problematik zu umgehen werden sowohl die Koordinaten, welche in den cpdb-Dateien zu finden sind, als auch die Koordinatenwerte aus den car_detected-Dateien bereits beim Einlesen als `double`-Werte gespeichert und auf fünf Nachkommastellen gerundet. Dadurch ge-

hen keine wertvollen Informationen verloren und die Werte der Zahlen sind auch für den Algorithmus gleich.

Wenn somit beide Autos eines Tracking-Ereignisses gefunden werden, wird dem Auto der Position 1 sein Nachfolger (`predecessor`) als `pointer` zu dem Car-Objekt der Position 2 hinterlegt. Dem Auto der zweiten Position wird umgekehrt auch sein Vorgänger (`successor`) als `pointer` hinzugefügt.

### 4.1.4 Import der Straßendaten

Um den Datensatz der Straßendaten sinnvoll einlesen zu können, wurde für diese ebenfalls eine eigene Klasse erzeugt. Dabei wurde sich dafür entschieden, die Daten möglichst nah an der gegebenen Struktur (vgl. Kapitel 2.3.4) zu modellieren.

#### Aufbau der Klasse StreetNode

Die Klasse stellt die Straßenstützpunkte dar, wie sie in der Datei vorkommen. Dabei werden die wichtigsten Daten wie folgt übernommen: Die `Navteq_ID`, die Reihenfolge des Punktes innerhalb des Straßenabschnitts und seine Koordinaten werden abgespeichert. Ebenso werden die Straßenkategorie und die Fahrspurkategorie hinterlegt - wobei letztere bisher nicht verwendet wird und nur zur Vollständigkeit der Daten übernommen wurde. Eine Übersicht über die Klasse ist im Klassendiagramm 4.3 gegeben.

Diese Klasse besitzt, wie die Klasse Car, mehrere Konstruktoren und eine Methode `convertStreetNodeToString()` um ein Objekt und seine Attribute als `string` zurückzugeben.

Außerdem ist es für die Sortierung und das Entfernen von Duplikaten ebenfalls nötig, den Operator < zu überladen. Dieser ist für die Klasse StreetNode wie folgt aufgebaut: In erster Linie soll die `Navteq_ID` zweier StreetNode-Objekte $a$ und $b$ entscheiden, welches kleiner ist:

$$a < b \ wenn\, gilt: \ a.Navteq\_ID < b.Navteq\_ID$$

Sollten beide Objekte dieselbe `Navteq_ID` besitzen, wird das kleinere Objekt über die Reihenfolge `section_order` bestimmt. Dabei muss allerdings ausgeschlossen werden, dass die betrachteten Punkte nicht tatsächlich gleich sind. Bedingt durch die Entstehung der Dateien (vgl. Kapitel 2.3.4) kann ein Stützpunkt in mehreren Dateien gleichzeitig vorliegen. Dieser besitzt dann jedoch unterschiedliche Werte für die `section_order`, da diese beim Erzeugen für jede Datei neu gezählt wird. Somit wird bei gleicher `Navteq_ID` nach folgenden Kriterien entschieden:

$$\begin{aligned} a < b \ wenn\, gilt: \ & a.Navteq\_ID = b.Navteq\_ID \\ & a.UTM\_coord \neq b.UTM\_coord \\ & a.Section\_order < b.Section\_order \end{aligned}$$

Hierbei sollte angemerkt werden, dass zwei StreetNode-Objekte, welche unterschiedliche `Navteq_IDs` aber dieselben Koordinaten besitzen, nicht gleichzusetzen sind, da diese „doppelten“ Punkte als Verknüpfung zwischen den Straßenabschnitten dienen.

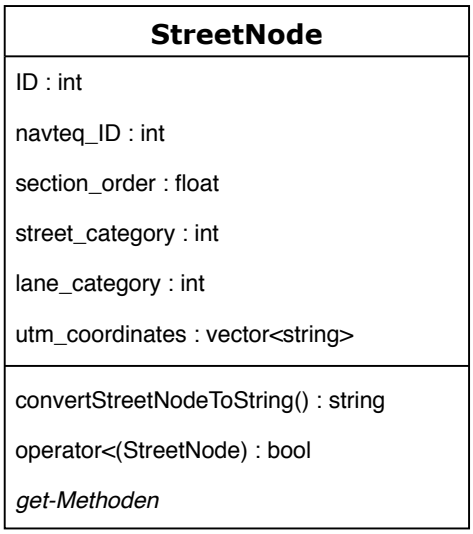



Abbildung 4.3: Klassendiagramm der Klasse StreetNode

**Einlesen der StreetNode Objekte**

Das Verarbeiten der navteq_roads-Dateien ist in der Datei „import_street_data.cpp“ implementiert und verläuft grundsätzlich analog zu den vorherigen Kapiteln: Mithilfe der „import_tools“ werden alle Dateien im übergebenen Ordner und den Unterordnern gefunden und Zeile für Zeile eingelesen. Aus dem Inhalt jeder Zeile wird jeweils ein StreetNode-Objekt erzeugt, welches zunächst einem `set` aller StreetNode-Objekte hinzugefügt wird. Da es beim Einlesen der Straßenstützpunkte häufig zu Duplikaten kommt, ist es sinnvoll nur die Speicherung von neuen Objekten zu erlauben.

Wie in Kapitel 2.3.4 beschrieben, werden hier nur Straßenabschnitte einer bestimmten Kategorie benötigt. Daher wurde eine Funktion entworfen, um die eingelesenen Objekte zu filtern und nur noch die benötigten Straßenabschnitte zu speichern.

Eine solche Funktion ist sehr einfach über das sogenannte `erase-remove` Idiom umzusetzen: Dies ist eine Technik in C++ um Objekte eines Containers (bspw. eines `vectors`) zu löschen, welche bestimmte Kriterien erfüllen. (Meyers 2001, S.139ff)

Das Kriterium der Straßenkategorie wurde dabei durch einen `Lambda`-Ausdruck umgesetzt.

Zuletzt gibt es auch für die Straßenstützpunkte eine Funktion, um einen `vector` von StreetNode-Objekten mithilfe der `convertStreetNodeToString()`-Methode der Klasse in eine CSV-Datei auszugeben.

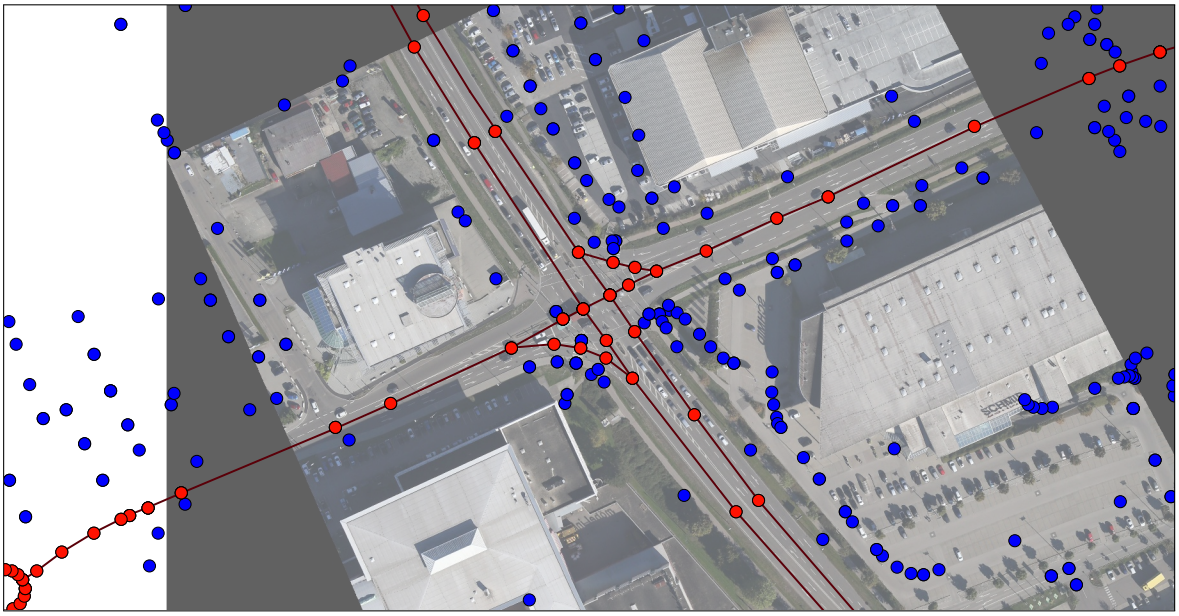


Abbildung 4.4: Straßenstützpunkte nach Kategorie gefiltert

Abbildung 4.4 zeigt die importierten Straßenstützpunkte der Kategorie 9 als rote Punkte. Als Referenz sind die Stützpunkte durch rote Linien verbunden und die eingelesenen Nodes aller Kategorien blau dargestellt.
Es ist gut zu erkennen, dass für eine der beiden Straßen der Kreuzung die Mittelachsen getrennt nach Fahrtrichtung bekannt sind, während die andere Straße nur eine allgemeine Mittelachse besitzt. Außerdem liegen für zwei der vier möglichen Rechtsabbiegespuren Stützpunkte vor.

## 4.2 Filtern der Daten

Die Daten der Verkehrserfassung enthalten zwar viele Informationen, allerdings sind nicht alle für diese Arbeit nötig. Zusätzlich kommt es in den Daten auch zu Fehldetektionen, welche die Ergebnisse negativ beeinflussen können. Daher ist es sinnvoll die importierten Daten zunächst zu filtern und zu bereinigen. In diesem Kapitel wird erläutert, welche Filterungen aus welchen Gründen erfolgen.

### 4.2.1 Kreuzungsbereich filtern

Die Luftbilder der Überfliegung von Senden umfassen einen großen Bereich um die gewählte Kreuzung. In diesem Bereich wurde durch die Verkehrserfassung ein großer Teil der Autos erfasst und somit auch in das Programm importiert. Wie in Abbildung 4.5 zu erkennen ist, stellt der zu untersuchende Kreuzungsbereich nur einen Bruchteil des erfassten Bereiches dar, weshalb eine Beschränkung auf die Kreuzung erfolgt.

Als Voraussetzung dafür muss der ungefähre Kreuzungsmittelpunkt vom Nutzer angegeben werden. Er kann ihn entweder manuell anhand von Kartendaten bzw. Luftbildern ermitteln oder ihn mithilfe eines Python Skriptes finden:
Dieses nutzt die Tatsache, dass sich eine Kreuzung in den navteq_roads-Daten durch den Übergang von vier verschiedenen Straßenabschnitten zeigt. Dadurch kommt ein Schnittpunkt der Abschnitte in den eingelesenen StreetNode-Daten ebenfalls vier mal - mit je unterschiedlichen `Navteq_IDs` - vor. Wenn die Straßenabschnitte in ca. 90° zueinander stehen wird angenommen, dass es sich hierbei um einen Kreuzungsmittelpunkt handelt.

Ist der Mittelpunkt bekannt, kann der Kreuzungsbereich in Form einer Bounding-Box

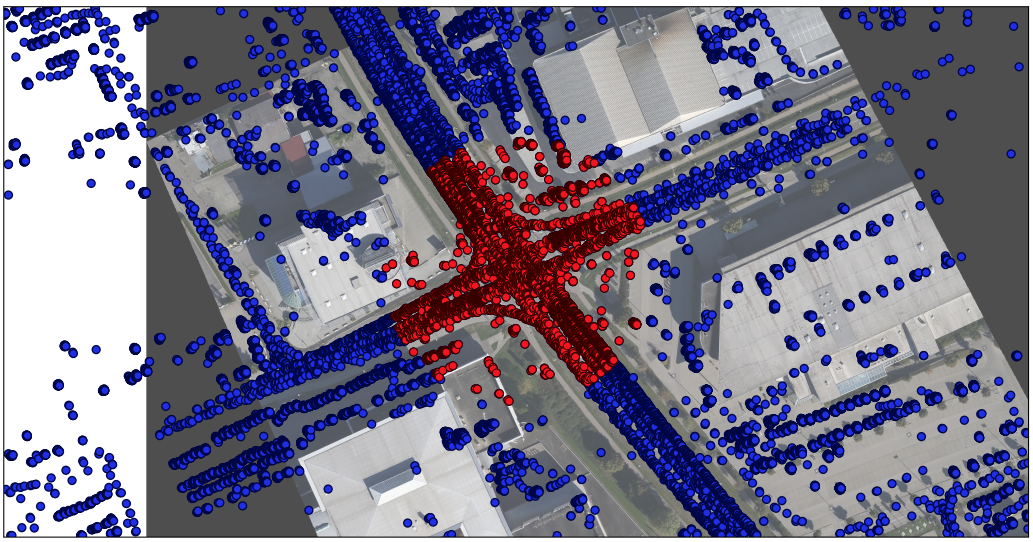


Abbildung 4.5: Autos im Kreuzungsbereich

bestimmt werden. Da eine solche Box allerdings nach den Himmelrichtungen ausgerichtet wäre und damit die Geometrie der Kreuzung schlecht erfasst werden kann, wurde sich hier für einen kreisförmigen Bereich entschieden. Der Radius dieses Kreises wurde für diesen Datensatz auf 60m festgelegt. Für jedes Auto des Car-`vectors` wird seine Distanz zu dem angegebenen Kreuzungsmittelpunkt berechnet und geprüft, ob diese kleiner als der Kreuzungsradius ist.

Um diese und noch folgende Berechnungen der analytischen Geometrie in C++ zu realisieren, wird die open-source Bibliothek „Eigen“ verwendet (*Eigen3 Dokumentation* 2020). Sie enthält unter anderem eigene Datentypen für mathematische Vektoren sowie Methoden um mit diesen zu rechnen. Die Berechnung der Distanz ist damit auf die Differenz der mathematischen Vektoren der beiden Punkte zurückführen. Mithilfe des `erase-remove` Idioms lassen sich hier alle Objekte des Car-`vectors` löschen, welche sich nicht innerhalb des Radius befinden. Abbildung 4.5 zeigt einen Ausschnitt aller importierten Daten (blaue Punkte). Die gefilterten Autos innerhalb des Kreuzungsbereichs sind rot dargestellt.

### 4.2.2 Autos nach Straße filtern

Ziel dieser Filterung ist es, nur noch diejenigen Autos zu betrachten, die sich auf den relevanten Straßen befinden (vgl. Kapitel 2.3.4). Da vor allem die Daten aus dem Deep Learning Algorithmus auch viele Autos abseits der Straßen - beispielsweise auf Parkplätzen - beinhalten, ist es sinnvoll, diese Detektionen herauszufiltern. Auch Autos, die sich auf Nebenstraßen befinden, sind in diesem Projekt nicht weiter wichtig.
Um diese Filterung durchführen zu können, muss zunächst ermittelt werden, auf welcher Straße sich ein Auto befindet. Dafür kann der Abstand eines Autos zu dem Straßenabschnitt der ihm am nächsten ist bestimmt werden und durch einen Schwellwert geprüft werden ob, bzw. auf welcher Straße sich das Auto befindet. Dies wurde wie folgt realisiert:

#### Zuordnung der Straßen

Die Funktion `getDistanceCar2Road()` liefert den Abstand zwischen einem Auto und einem Straßenabschnitt, welcher über zwei StreetNode-Objekte definiert ist. Mathematisch lässt sich dies auf die Berechnung der Distanz zwischen einem Punkt und einer Linie zurückführen. In der analytischen Geometrie kann dafür die Orthogonalprojektion eines Punktes auf eine Gerade berechnet werden (Wikipedia 2020). Das Ergebnis ist der sogenannte Lotfußpunkt als Vektor.
Abbildung 4.6 zeigt die Orthogonalprojektion $\vec{p_{Lot}}$ des Autos $p_{car}$ auf die Gerade durch $\vec{p_1}$ und $\vec{p_2}$. Die Berechnung des Lotpunktes $\vec{p_{Lot}}$ als Vektor erfolgt durch:

$$\vec{p_{Lot}} = \vec{p_1} + \lambda \cdot (\vec{p_2} - \vec{p_1}) \qquad mit \qquad \lambda = \frac{(\vec{p_{car}} - \vec{p_1}) \cdot (\vec{p_2} - \vec{p_1})}{\|\vec{p_2} - \vec{p_1}\|^2}$$

Der Abstand zwischen dem Auto $\vec{p_{car}}$ und dem berechneten Lotfußpunkt $\vec{p_{Lot}}$ lässt sich anschließend über die Differenz ihrer Vektoren berechnen:

$$d = \|\vec{p_{car}} - \vec{p_{Lot}}\|$$

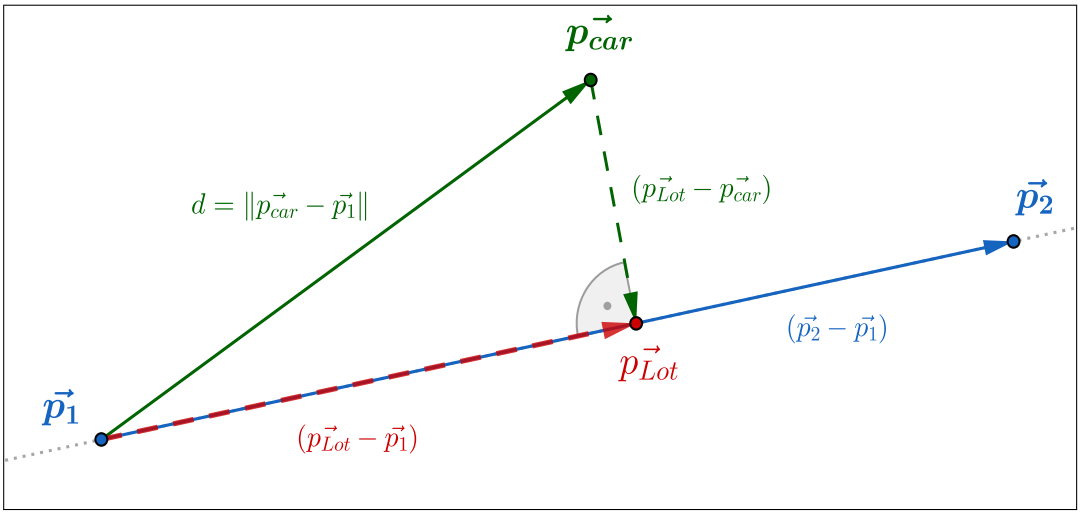



Abbildung 4.6: Berechnung Orthogonalprojektion

Da diese Berechnung von einer endlosen Geraden ausgeht, ein Straßenabschnitt allerdings nur zwischen den beiden Stützpunkten verläuft, kann es zu Problemen kommen, wenn sich das Auto nicht parallel zum Straßenabschnitt befindet.
Dieser Fall ist in Abbildung 4.7 mit dem Auto $\vec{p_{car_2}}$ in orange dargestellt. Über die Abstände $l$, $d_1$ und $d_2$ lässt sich einfach prüfen, ob der Lotfußpunkt $\vec{p_{Lot}}$ eines Autos innerhalb des Straßenabschnittes liegt. Ist dies nicht der Fall - vgl. $\vec{p_{car_2}}$ aus Abbildung 4.7 - wird als Abstand des Autos zum Straßenabschnitt die Distanz zwischen dem Auto und dem nächsten der beiden Stützpunkte ($\vec{p_1}$ oder $\vec{p_2}$) festgelegt.

Die Implementierung dieser Berechnungen wurde auch hier durch die Funktionen der Bibliothek „Eigen“ erleichtert. Mithilfe dem Erzeugen mathematischer Vektoren aus

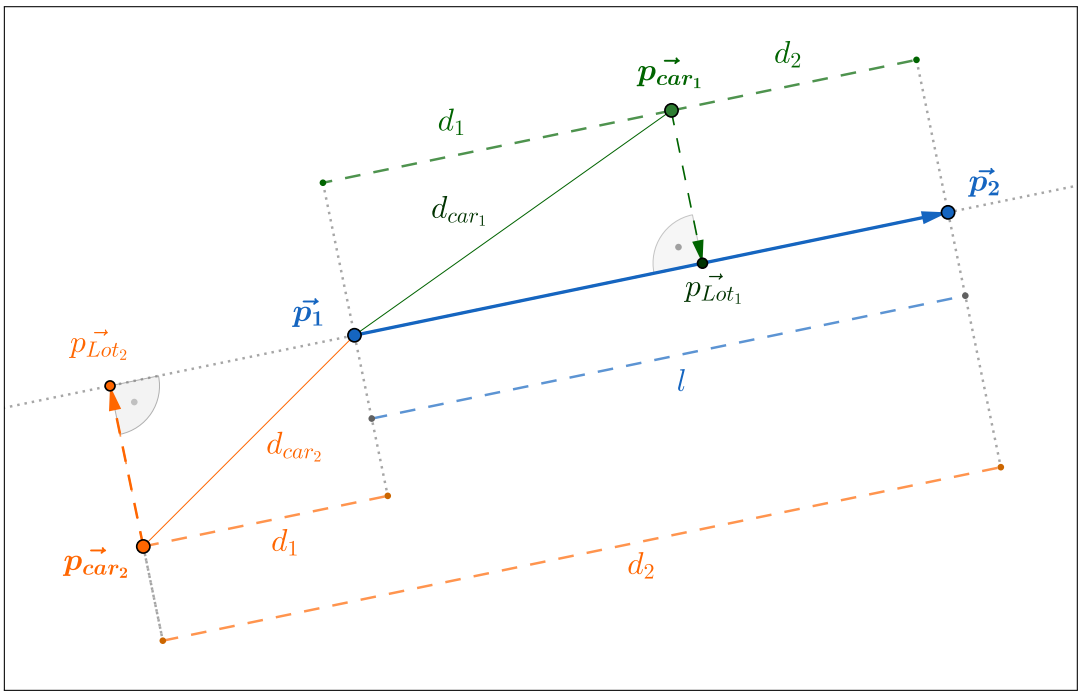



Abbildung 4.7: Orthogonalprojektion Sonderfall

dem Car-Objekt und den beiden StreetNode-Objekten lassen sich die obigen Formeln direkt implementieren. Für das Skalarprodukt von Vektoren und die Normierung konnten die vorhandenen Funktionen `dot()` und `norm()` verwendet werden.

Um die nächste Straße zu einem gegebenen Auto zu finden, müssen die Distanzen zu allen Straßenabschnitten berechnet werden und das Minimum gefunden werden. Dazu nimmt die Funktion `findClosestStreet()` ein Car-Objekt und einen sortierten `vector<StreetNode>` mit allen Straßenstützpunkten entgegen. Anschließend wird die Funktion `getDistanceCar2Road()` mit jedem Paar von zusammenhängenden StreetNode-Objekten aufgerufen und das Minimum der berechneten Distanzen bestimmt. Damit ist der nächste Straßenabschnitt des Autos gefunden und den Attributen `Navteq_ID` und `street_dist` des Autos werden die entsprechenden Werte zugewiesen.

### Filterung der Autos

Sobald in den Attributen aller Autos gespeichert ist, wie groß die Distanz zu ihrem nächsten Straßenabschnitt ist, kann mithilfe eines Schwellwertes gefiltert werden: Dazu wird ein `Lambda`-Ausdruck verwendet, welcher für ein übergebenes Auto prüft, ob sein Wert für `street_dist` größer als der festgelegte Schwellwert ist. Mit diesem Entscheidungskriterium wird erneut das `erase-remove` Idiom angewandt und somit alle Car-Objekte, die den Schwellwert überschreiten, aus dem `vector` gelöscht.

Anhand des Datensatzes zeigt sich, dass die Wahl des maximalen Abstands zwischen Auto und Straßenabschnitt auch stark von der Vollständigkeit der Straßendaten abhängt: Wie in Kapitel 4.1.4 gezeigt, werden Straßen nur durch ihre Mittelachse dargestellt. Zusätzlich sind hier die äußersten Abbiegespuren teilweise nicht vorhanden, weshalb der gemessene Abstand der dort abbiegenden Autos zum nächsten Straßenabschnitt entsprechend hoch (ca. 15m) ist. Das heißt der maximale Straßenabstand muss im schlechtesten Fall auch den Durchmesser der Kreuzung berücksichtigen. Dies führt - wie in Abbildung 4.8 zu sehen ist - dazu, dass auch Autos, die am Rand der Straße stehen, in die gefilterten Autos übernommen werden. Im Falle von parkenden Autos können diese „falschen“ Ergebnisse allerdings mit dem Algorithmus des nächsten Kapitels automatisch gefiltert werden.

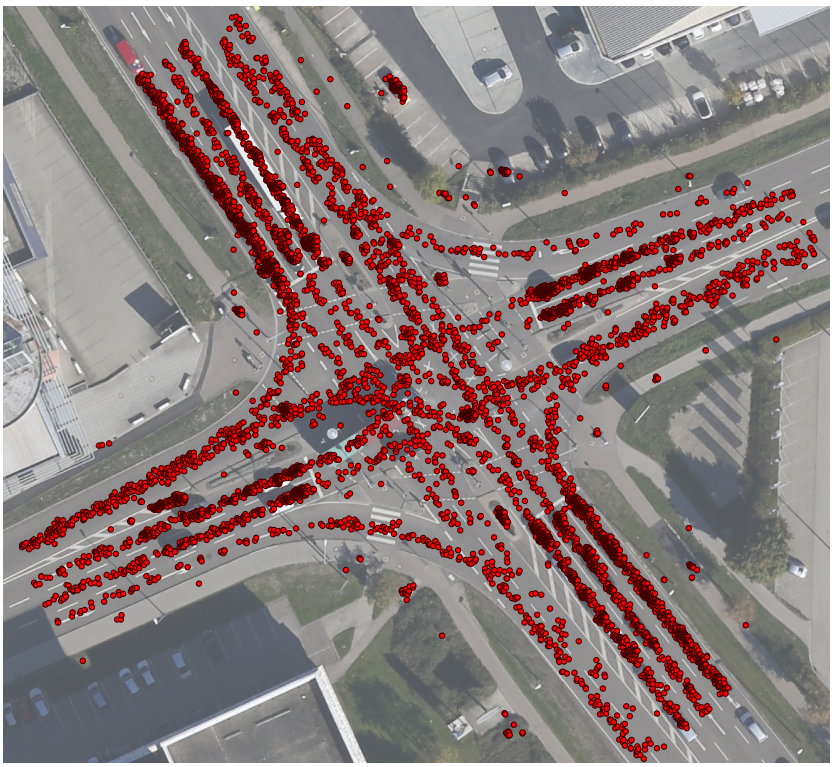


Abbildung 4.8: Ergebnis der Filterung nach Straßenabstand

### 4.2.3 Fehldetektionen entfernen

Fehlerhafte Detektionen von Autos können in keinem der Detektions-Algorithmen ausgeschlossen werden. Daher ist es sinnvoll, die Auswirkungen der schwerwiegensten Fehler zu minimieren. Ein solches Problem sind die dichten Ansammlungen (Cluster) von Fehldetektionen. Diese treten vor allem in dem Deep Learning Algorithmus auf, da hier gelegentlich Muster in den Luftbildern (beispielsweise Schatten einer Ampel) fälschlicherweise als Autos identifiziert wurden (vgl. Kapitel 2.3.1). Sind diese dann in vielen Bildern zu sehen entstehen Ansammlungen von vielen Detektionen auf engen Raum. Der Ausschnitt in Abbildung 4.9 zeigt ein paar dieser Cluster. Wie im Ausschnitt zusätzlich zu erkennen ist treten diese Cluster auch durch die stehenden Autos vor Ampeln auf. Diese Daten stellen nicht unbedingt Fehldetektionen dar, könnten sich aber dennoch negativ auf das Endergebnis auswirken: Bei einer Mitte-

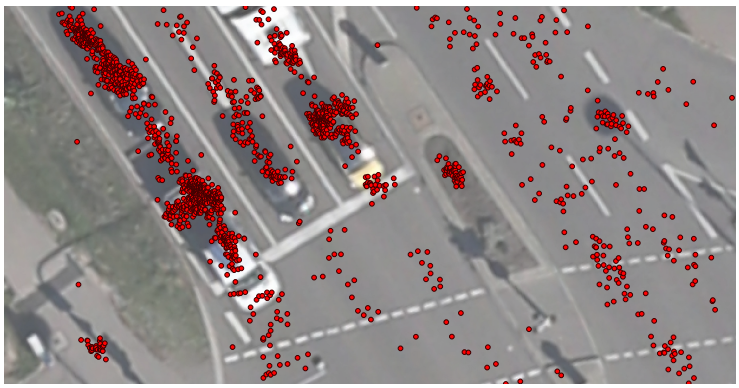


Abbildung 4.9: Ausschnitt der Detektionen mit Clustern

lung aller Daten einer Fahrspur würden solche Bereiche aufgrund der hohen Anzahl an Punkten stärker gewichtet werden, wodurch der Gesamtverlauf einer gemittelten Trajektorie beeinflusst werden kann.

Um diese Cluster zu finden, wird in der Funktion `removeClusteredDetections()` für jedes Auto einzeln geprüft, ob es sich in einem Cluster befindet. Dafür werden die Abstände zu den übrigen Autos berechnet und im Falle einer Unterschreitung eines maximalen Radius ein Zähler erhöht. Dieser zeigt wieviele Autos sich im direkten Umfeld des betrachteten Autos befinden und ist somit Hinweis auf ein Cluster. Wurden alle Autos geprüft und ist der Zähler kleiner als der übergebene Schwellwert für die Entscheidung zu einem Cluster, zählt das Auto nicht zu einem Cluster und wird einem neuen `vector` für die bereinigten Car-Objekte hinzugefügt. Dieser `vector` wird am Ende der Funktion zurückgegeben. Das Ergebnis ist in den Abbildungen 4.10 und 4.11 visualisiert: Die Autos des bereinigten `vectors` sind in Gelb, die entfernten Cluster in Rot dargestellt. Zu sehen ist, dass die meisten Cluster erkannt und erfolgreich entfernt wurden - wobei an manchen Stellen einzelne Fehldetektionen an den Rändern der Cluster bestehen bleiben. Die entstandenen Lücken vor den Ampeln sind dabei nicht vermeidbar und könnten bei größeren Clustern zu Problemen führen. Eine mögliche Lösung hierfür wäre eine Erkennung von Fahrzeugen, die an Ampeln

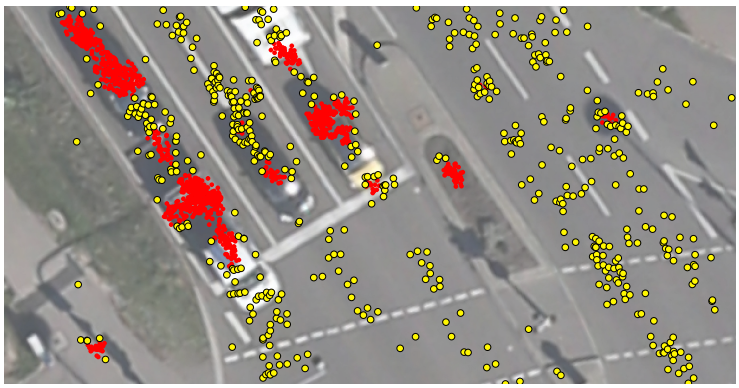


Abbildung 4.10: Entfernte Cluster im Ausschnitt

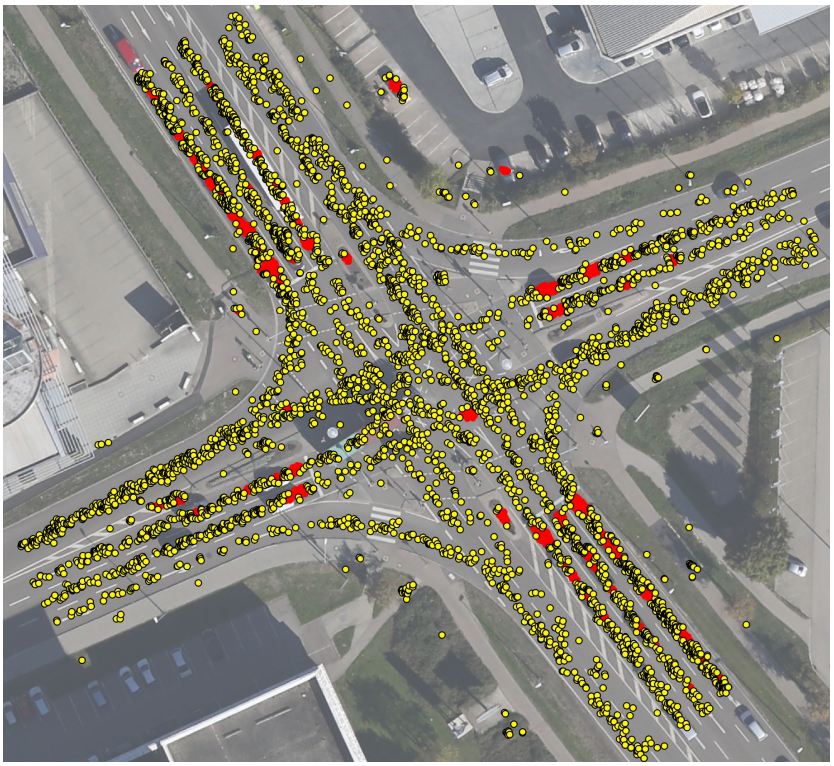


Abbildung 4.11: Entfernte Cluster im Kreuzungsbereich

halten. Dies könnte durch einen Fuzzy-Logik-Algorithmus aus (Knöttner u. a. 2019) umgesetzt werden.

Das Ergebnis dieser drei Filtervorgänge ist ein `vector` mit allen übrig gebliebenen Car-Objekten. Dieser wird nun verwendet, um eine automatisierte Fahrspurerkennung zu implementieren.

## 4.3 Erkennung von Fahrspuren

Um die optimale Trajektorie einer Fahrspur aus allen Auto-Detektionen erhalten zu können, muss zuerst bekannt sein, welche Detektionen zu dieser Fahrspur gehören. Dies kann vor allem im Kreuzungszentrum schwer zu entscheiden sein, da sich hier die Detektionen aller Fahrspuren treffen und somit ein Chaos verschiedener Autos mit unterschiedlichen Ausrichtungen entsteht. Dieses ist in Abbildung 4.12 durch eine Vektordarstellung der Autos nach ihrer Ausrichtung dargestellt. Eine Möglichkeit

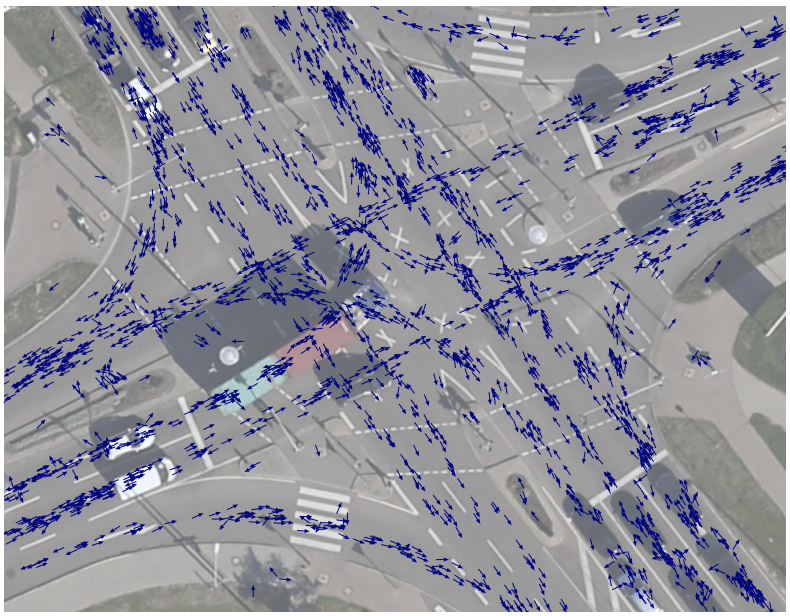


Abbildung 4.12: Kreuzungsmitte mit Vektoren der Autos

der Gruppierung nach Fahrspuren ist es, den Kreuzungsbereich vom Rand beginnend nach Innen zu erfassen und dabei Fahrspuren „nachzuverfolgen“. Die Grundlage dafür wird durch eine Nachfolgersuche geschaffen: Für jedes Auto werden seine Nachfolger, unter Berücksichtigung der Ausrichtung und Position der Autos, ermittelt. Somit können durch ein paar bekannte „Startpunkte“ am Rand der Kreuzung Pfade für die Fahrspuren erzeugt werden.

Das genaue Vorgehen wird in den folgenden Abschnitten erläutert.

Da in dem bereinigten `vector` der Car-Objekte alle relevanten Autos vertreten sind, soll dieser für den weiteren Verlauf als Basis dienen. Um zu verhindern, dass Elemente aus diesem entfernt oder hinzugefügt werden, kann ein `vector` aus sogenannten `pointern` zu den Car-Objekten erstellt werden.
`pointer` beinhalten die Speicheradresse des Objekts dem sie zugeordnet wurden. Durch `pointer` kann im Programm direkt auf das Objekt zugegriffen werden. Dies ist hier vor allem nützlich um Car-`vectoren` zu erstellen, die nur einen Ausschnitt aller Objekte - wie beispielsweise Autos einer gewählten Fahrspur - beinhalten.
Statt solche `vectoren` durch das Kopieren der benötigten Objekte zu erzeugen, wird ein `vector` mit Car-`pointern` erstellt, welcher lediglich die `pointer` zu den entsprechenden Objekten enthält. Dadurch wird nicht nur der Speicherbedarf zur Laufzeit des Programmes verringert, sondern es wird auch sichergestellt, dass mögliche Änderungen an Objekten des Ausschnittes auch für die eigentlichen Objekte des bereinigten Car-`vectors` gelten.

### 4.3.1 Finden von Randpunkten

Um diejenigen Autos auszuwählen, die sich am Rand des definierten Kreuzungsbereiches befinden, wird der Abstand der Autos zum Kreuzungsmittelpunkt betrachtet. Aufgrund der vorangegangenen Filterung des kreisförmigen Kreuzungsbereichs entspricht dieser Abstand maximal dem Radius des Kreuzungsbereichs (hier: 60m).
Als Randpunkte werden solche Autos definiert, die sich in einem bestimmten Abstandsbereich befinden: beispielsweise alle Autos mit einem Abstand zwischen 50m und der maximalen Distanz von 60m. Um diesen Bereich im Code nicht als absoluten Wert anzugeben, wird stattdessen ein prozentualer Anteil des Kreuzungsradius verwendet.

Die Funktion `getCarsOnBorderCircle()` nimmt den `pointer-vector` der bereinigten Autos und den prozentualen Wert $p$ entgegen. Es werden diejenigen Autos bestimmt, für deren Abstand $d$ zum Kreuzungmittelpunkt gilt:

$$R \cdot p \leq d \leq R$$

Wobei $R$ den Radius des Kreuzungsbereiches darstellt. Für $p$ hat sich gezeigt, dass der Wert 0,7 - also 70% - gut geeignet ist.

Als Startpunkte der Fahrspursuche sollen nur Autos gelten, die sich auch auf dem Weg in die Kreuzung befinden. Da die Verkehrserfassung neben den Koordinaten auch die Ausrichtung eines Autos angibt, lässt sich mit dieser prüfen, ob ein Auto in den Kreuzungsbereich hineinfährt oder diesen verlässt. Diese Prüfung wird in der Funktion `removeOutgoingCars()` über das `erase-remove` Idiom durchgeführt. In einem `Lambda`-Ausdruck wird darin die Funktion `IS_CAR_FACING_INTERSECTION()` aufgerufen, welche prüft ob das jeweilige Auto zum Mittelpunkt ausgerichtet ist. Dies geschieht indem der Richtungswinkel des Autos zum Kreuzungmittelpunkt berechnet und mit dem Winkel der Ausrichtung verglichen wird. Liegt die Differenz bei weniger als 20gon ist die Bedingung erfüllt und das Auto zählt zu den Startpunkten. Sind somit alle Startpunkte ermittelt wird ein `vector` mit den `pointern` der gewählten Autos zurückgegeben.

Zuletzt erfolgt noch eine Gruppierung der Startpunkte nach ihren Straßenabschnitten über die Funktion `groupCarsByNavteqIDAndStreetDist()`. Da in den Attributen jedes Autos schon sein nächster Straßenabschnitt (als `Navteq_ID`) und die Distanz zu diesem gespeichert ist, werden in dieser Funktion lediglich die vorkommenden `Navteq_IDs` ermittelt und Autos derselben `Navteq_ID` gruppiert.
Eine erneute Filterung nach der Distanz zum Straßenabschnitt kann die Ergebnis-

se verbessern: der Schwellwert, der in Kapitel 4.2.2 gewählt werden musste, kann im Bereich der Startpunkte kleiner gewählt werden, da sich die Autos hier noch auf geraden Fahrspuren befinden.

Mit einem neuen Schwellwert von 10m ergeben sich die Startpunkte aus Abbildung 4.20: Die schwarz dargestellten Punkte entsprechen allen Randpunkten - unabhängig ihrer Ausrichtung. Die zur Kreuzung ausgerichteten Autos sind in die Startpunktgruppen aufgeteilt und farblich getrennt dargestellt.
Es fällt auf, dass auch auf den Spuren, welche die Kreuzung verlassen, Startpunkte liegen. Dies ist auf eine fehlerhafte Erkennung der Ausrichtung zurückzuführen. In der Fahrspursuche könnten diese zu Problemen führen und sollten daher berücksichtigt werden.

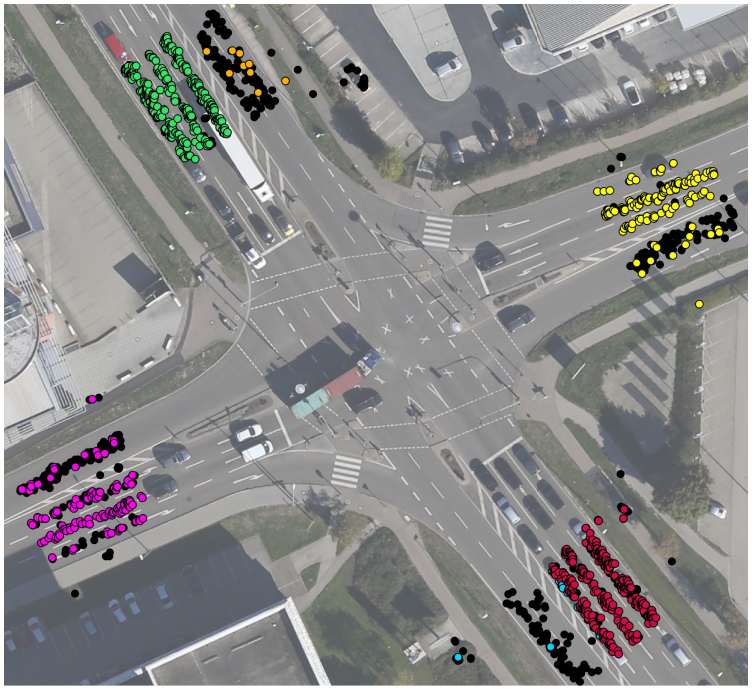


Abbildung 4.13: Randpunkte und Startpunktgruppen

### 4.3.2 Finden von Nachfolgern eines Autos

Die Aufgabe Nachfolger eines Autos zu bestimmen ist essentiell für alle weiteren Algorithmen dieses Projekts. Die Idee ist es, zu einem gewählten Auto („Startauto“) alle Autos („Nachfolger“) zu finden, welche seinen weiteren Fahrtverlauf darstellen könnten. Dabei sind drei Größen entscheidend, um ein Auto als Nachfolger zu identifizieren:

- $d$: die Distanz zwischen den beiden Autos
- $\alpha$: der Winkel des Startautos zu dem Nachfolgerauto
- $\Delta riwi$: die Differenz der Ausrichtungen ($riwi$) der beiden Autos

Diese Größen sind in Abbildung 4.14 am Beispiel von zwei Car-Objekten (schwarze Punkte mit Vektoren ihrer Ausrichtung) dargestellt.

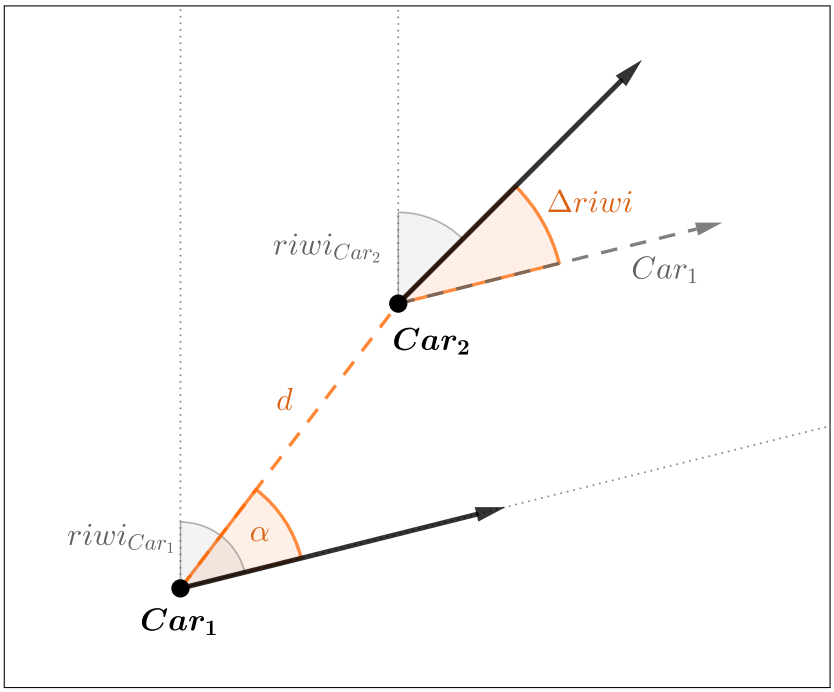



Abbildung 4.14: Parameter der Nachfolgersuche

Für die Parameter müssen geeignete Grenzwerte gewählt werden, nach denen entschieden wird, ob ein Auto als Nachfolger gilt. Die Funktion `findNextCars()` nimmt den `pointer` des Startautos, die Grenzwerte der drei Parameter und den `vector` aller Car-Objekte entgegen. Durch eine `for`-Schleife wird jedes Auto als möglicher Nachfolger-Kandidat geprüft und die Parameter über Differenzbildung berechnet. Liegen die Ergebnisse innerhalb der Grenzwerte, wird dem Attribut `successors` des Startautos der `pointer` des neu gefundenen Nachfolger-Objekt hinzugefügt. Gleichzeitig wird auch das Attribut `predecessors` des Nachfolger Autos um den `pointer` zu dem Startauto erweitert. Dadurch ist die Beziehung zwischen den Autos in beide Richtungen (Vorgänger und Nachfolger) bekannt. Da dies für alle Autos wiederholt wird, besitzt das Startauto am Ende der Funktion meist viele `pointer` in seinem `successors`-Attribut. Falls kein einziger Nachfolger gefunden wurde bleibt der `vector` des Attributes leer.

Jedes Auto, für das die Nachfolgersuche erfolgreich war und dessen `successors`- oder `predecessors`-Attribut somit nicht leer ist, wird über seinen `pointer` für die weitere Verarbeitung gespeichert.

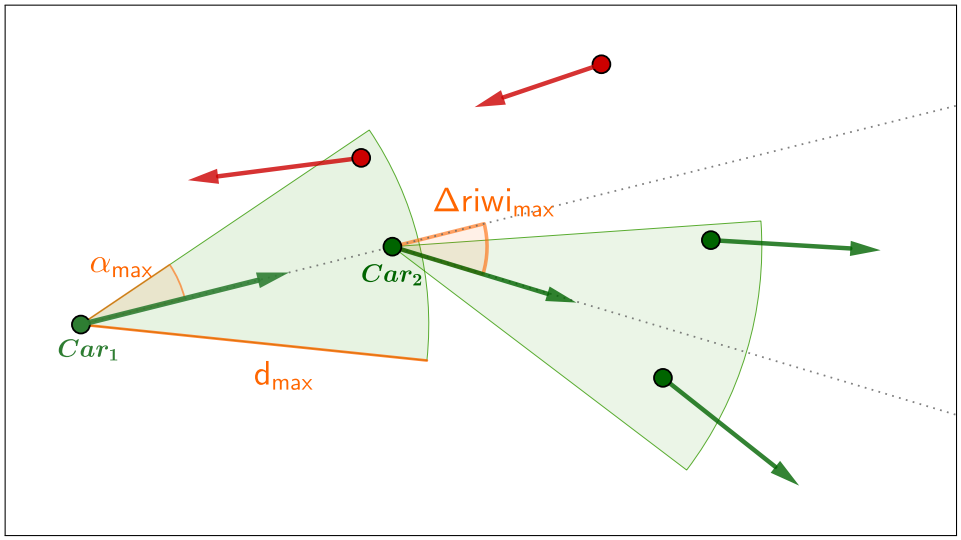



Abbildung 4.15: Beispiel der Nachfolgersuche

Dieser Ablauf ist in Abbildung 4.15 anhand eines Besipiels skizziert: Die Grenzwerte $d_{max}$ und $\alpha_{max}$ erzeugen einen Kreissektor in dem die Nachfolgerautos liegen müssen.
Zusätzlich darf auch die Differenz der Ausrichtungen nicht den Grenzwert $\Delta riwi_{max}$ überschreiten. Demnach besitzt das Auto $Car_1$ des Beispiels nur einen gültigen Nachfolger ($Car_2$). Für das Auto $Car_2$ können dagegen zwei Nachfolger gefunden werden.

Mit dem Datensatz dieser Arbeit wurden verschiedene Werte der drei Parameter getestet. Um Fehldetektionen und Spurwechsel nicht mit zu erfassen sollten die Werte so klein wie möglich gewählt werden. Sind sie zu klein werden auf Abbiegespuren keine Nachfolger erfasst, wodurch diese Fahrspuren wegfallen. Die besten Ergebnisse wurden mit den Werten: $d = 5m$, $\alpha = 30gon$ und $\Delta riwi = 30gon$ erzielt.

### 4.3.3 Ansatz aus der Graphentheorie

Diese Nachfolgersuche wird auf alle Autos des `vector<Car>` angewendet, sodass sämtliche Vorgänger- und Nachfolgerbeziehungen zwischen den Autos bekannt sind. Das bedeutet jedes Car-Objekt kennt (falls vorhanden) alle seine Vorgänger- und Nachfolger-Objekte. Damit lässt sich der Datensatz als symmetrischer gerichteter Graph verstehen: Dieser entspricht einem gerichteten Graphen, der zu jeder gerichteten Kante auch die gegengerichtete Kante kennt (Chen 1997, S.31).
Hier entsprechen die Car-Objekte den Knoten und die Vorgänger-/ Nachfolger-Beziehungen den gerichteten/ gegengerichteten Kanten. In Abbildung 4.16 sind die Vorgänger-/ Nachfolger-Beziehungen zwischen den Autos des Beispiels aus Abbildung 4.15 dargestellt.

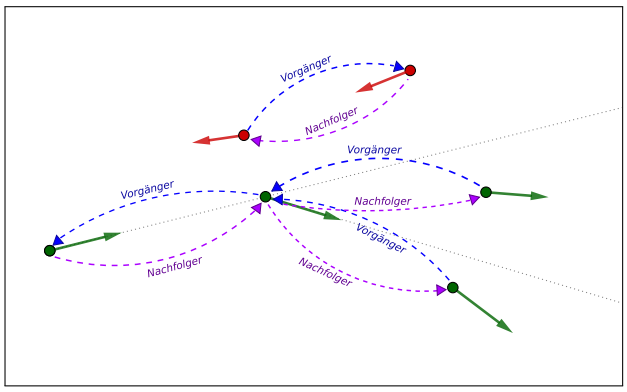



Abbildung 4.16: Darstellung der Vorgänger-Nachfolger-Beziehungen

### Die Breitensuche

Um Spannbäume für Fahrspuren zu erzeugen, können - beginnend von einem Startauto - iterativ alle verbundenen Knoten sowie deren verbundene Knoten betrachtet werden. Dabei muss darauf geachtet werden nur Kanten einer Richtung zu verfolgen - wie beispielsweise alle Nachfolger und auch deren Nachfolger. Dieses Vorgehen kann solange fortgeführt werden bis eine maximale Anzahl an Knoten erreicht wird oder keine weiteren Kanten mehr erreichbar sind. Das Prinzip dahinter ist die sogenannte Breitensuche (vgl. Abbildung 4.17). Diese beginnt mit einem Startknoten auf

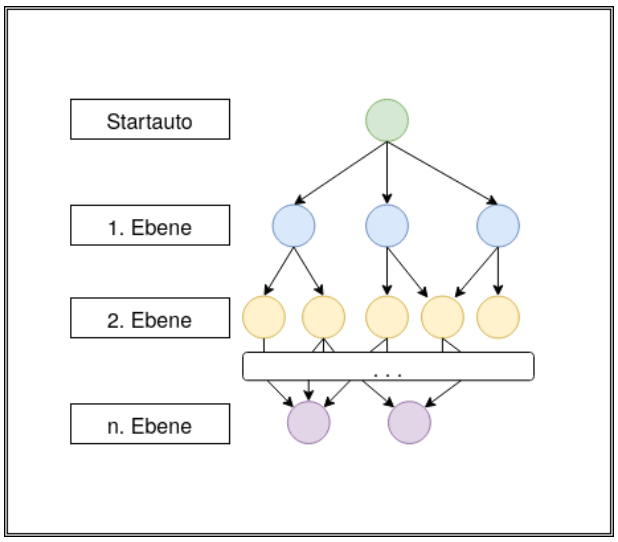



Abbildung 4.17: Schematische Darstellung der Breitensuche

Ebene 0. Alle Knoten, die über eine (gerichtete) Kante von diesem erreichbar sind, werden ermittelt und bilden die Ebene 1. Für jeden Knoten der Ebene 1 wird das Vorgehen analog zum Startknoten wiederholt, sodass die Knoten der Ebene 2 gefunden werden. Dies geschieht bis die n-te Ebene erreicht wird in der es keine neuen Kanten mehr gibt. (Turau 2009, S.121ff)

### Implementierung der Breitensuche

Der Ablauf der Breitensuche ist in der Funktion `findSpanningTree()` realisiert und wird im Flussdiagramm 4.19 dargestellt.
Dieser Funktion müssen ein `vector<Car*>` mit mindestens einem Startauto und ein `bool` (`SEARCH_BY_SUCCESSOR`) übergeben werden. Beginnend mit den Startknoten in `start_cars` werden alle ausgehenden Knoten ermittelt und die `pointer` zu diesen Car-Objekten in dem `vector temp_cars` gesammelt. Ob es sich bei den ausgehenden Knoten um die Vorgänger oder Nachfolger handelt, ist durch den übergebenen `bool SEARCH_BY_SUCCESSOR` angegeben. Das Ermitteln der neuen Car-Objekte kann dann durch eine entsprechende Abfrage des Attributs `predecessors` bzw. `successors` des Startautos erfolgen. Besitzt das aktuelle Auto keine Vorgänger bzw. Nachfolger wird es dem `vector end_cars_of_spanning_tree` angehängt. Falls jedoch Vorgänger bzw. Nachfolger vorhanden sind werden diese dem `vector temp_cars` hinzugefügt.
Nachdem dieser Vorgang für alle Elemente aus `start_cars` durchgeführt ist, wird der Inhalt des `vectors temp_cars`- nach dem Löschen von mehrfach vorkommenden Einträgen - in den übergreifenden `vector start_cars` eingefügt und ersetzt die vorherigen Werte. Die Objekte dieses `vectors` werden nun wieder als neue Startpunkte gewählt und das Speichern der Vorgänger bzw. Nachfolger wird wiederholt.
Dies geschieht bis keine neuen Vorgänger bzw. Nachfolger mehr gefunden werden können und der `vector start_cars` keine Elemente enthält. Das Ergebnis ist der

`vector all_cars_of_spanning_tree` mit `pointern` zu allen Car-Objekten sowie der `vector end_cars_of_spanning_tree` mit allen Endpunkten des Spannbaumes. Diese beiden Listen werden in einem `std::pair` zurückgegeben.

Nach einer Anwendung auf den Datensatz stellte sich heraus, dass es in der Funktion zu Endlosschleifen kommen konnte. Diese sind durch das Zusammenspiel der Nachfolgersuche aus Kapitel 4.3.2 und dem Einbinden der Tracking Informationen (vgl. Abschnitt „_cpdb-Dateien“ aus 4.1.3) entstanden: Es konnte dabei vorkommen, dass sich zwei Autos gegenseitig als Vorgänger und auch Nachfolger speichern. Dadurch findet die Breitensuche niemals Endpunkte, sondern wechselt in einer Endlosschleife zwischen den beiden Autos als `start_cars`. Um dies zu unterbinden wurde eine Abbruchbedingung eingebaut, die prüft ob sich die Anzahl der eindeutigen Elemente des Spannbaumes nach einem Durchgang noch ändert. Ist dies nicht der Fall, wurden in diesem Durchgang keine neuen Objekte hinzugefügt und die Schleife wird unterbrochen. Abbildung 4.18 zeigt einen Ausschnitt eines Spannbaums, der von einem Auto

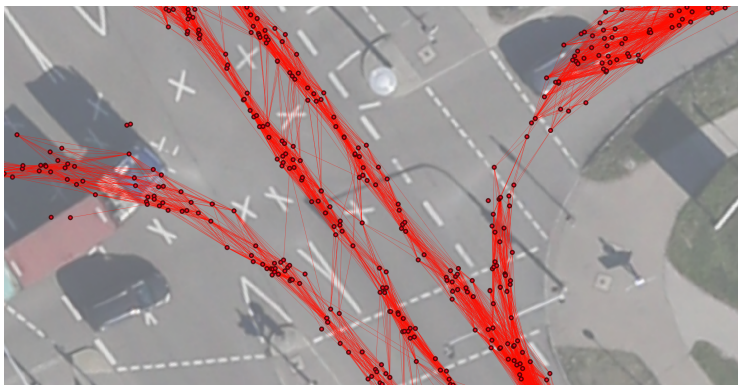


Abbildung 4.18: Spannbaum eines Fahrzeuges

am südlichen Rand der Kreuzung beginnt. Die Linien stellen dabei alle Nachfolger-Beziehungen dar. Es fällt auf, dass der Spannbaum alle drei Fahrspuren beinhaltet, die von diesem Randbereich der Kreuzung erreicht werden können. Für die einzelnen Fahrspuren wird dieser Spannbaum in Kapitel 4.3.5 genauer unterteilt.

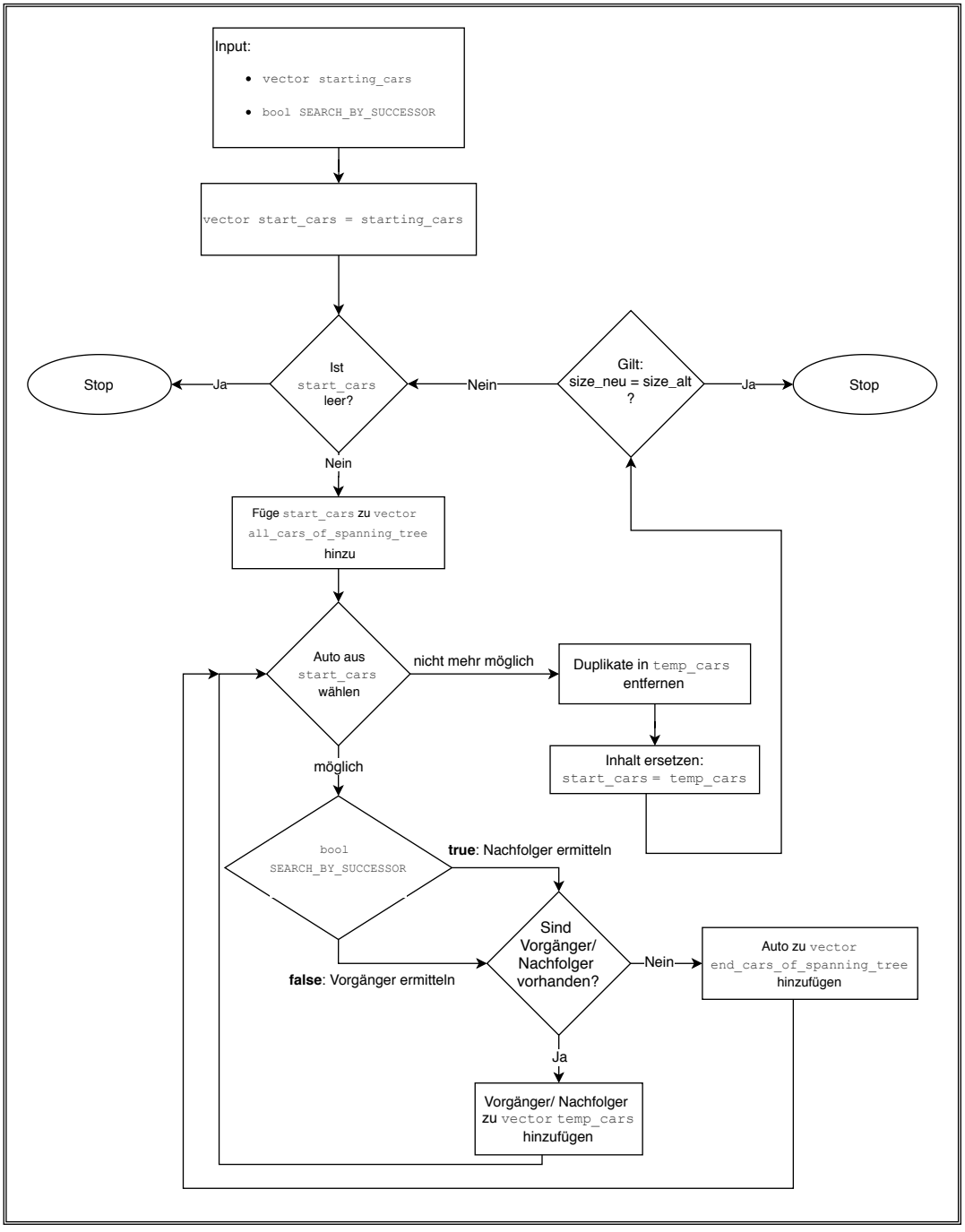
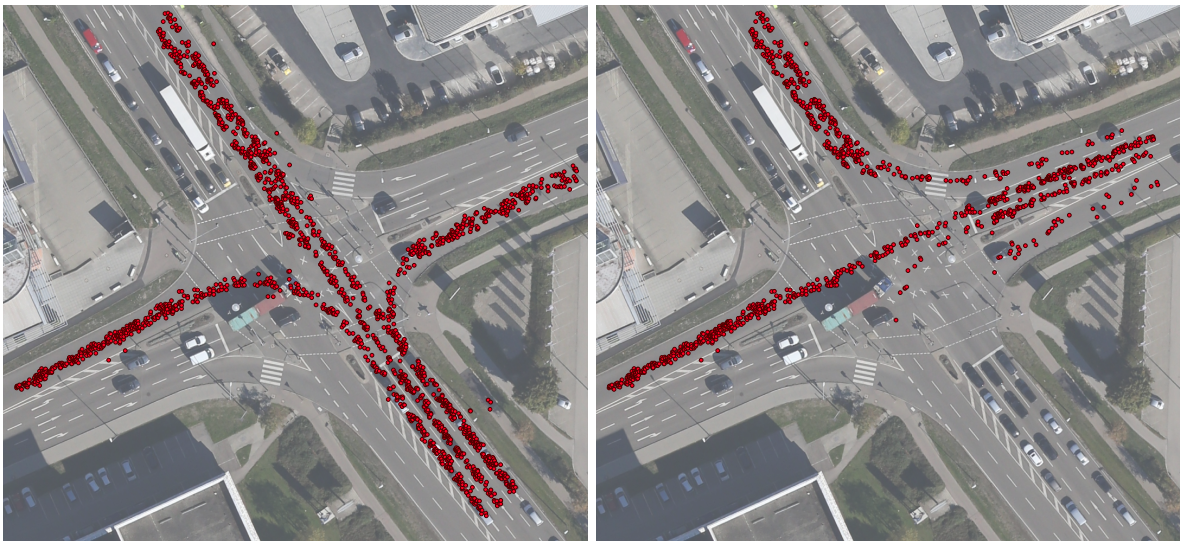



Abbildung 4.19: Flussdiagramm der Breitensuche

### 4.3.4 Finden von Fahrspurgruppen

Die Funktion `findSpanningTree()` kann nicht nur den Spannbaum eines einzelnen Startknotens bestimmen, sondern auch einen `vector` von Startpunkten entgegennehmen. Mit den ermittelten Randpunkten aus Kapitel 4.3.1 können dann Spannbäume von den verschiedenen Startpunktgruppen beginnend entstehen. Da sichergestellt wurde, dass die Startautos mit Fahrtrichtung in die Kreuzung stehen, ist davon auszugehen, dass sie sich auf einer der Fahrspuren zu der Kreuzung hin befinden. Sollte das nicht der Fall sein und die Fahrtrichtung fälschlicherweise zur Kreuzung zeigen, sollten Spannbäume die von diesen Autos beginnen nicht groß werden, da nicht davon auszugehen ist, dass es genug dieser Fehldetektionen gibt.

Für jede Startpunktgruppe lässt sich die Breitensuche über die Nachfolgerbeziehungen durchführen und somit einen Spannbaum berechnen, der alle Autos beinhaltet, die aus diesem Bereich in die Kreuzung gefahren sind. In Abbildung 4.20 sind die Ergebnisse von zwei der Startpunktgruppen in QGIS dargestellt:



(a) Fahrspurgruppe 2 (b) Fahrspurgruppe 1

Abbildung 4.20: Auswahl von Ergebnissen der Fahrspurgruppen

Der linke Teil zeigt die Autos der Fahrspurgruppe 2, die vollständig erfassst wurde. Im rechten Teil der Abbildung (Fahrspurgruppe 1) ist zu sehen, welche Auswirkungen die oben erwähnten Autos mit fehlerhaften Ausrichtungen haben können.

Zusätzlich wird hier ein Problem der Nachfolgersuche sichtbar: Die Linksabbiegespur ist nicht vollständig erfasst. Im Beobachtungszeitraum der Verkehrserfassung sind dort wenige Autos abbgebogen, was zu einer geringen Dichte an Detektionen geführt hat. Grundsätzlich sind zwar einige Punkte vorhanden, allerdings reichen die Parameter der Nachfolgersuche nicht aus, um die Lücken in den Detektionen zu schließen. Es wurden zwar auch größere Toleranzen bei den Parametern getestet, aber der Einfluss von falschen Ausrichtungen wurde dadurch so groß, dass falsche Fahrspuren entstehen konnten. Ein besserer Ansatz wäre es, solche Bereiche über längere Zeiträume oder zu unterschiedlichen Zeitpunkten zu beobachten.

### 4.3.5 Finden einzelner Fahrspuren

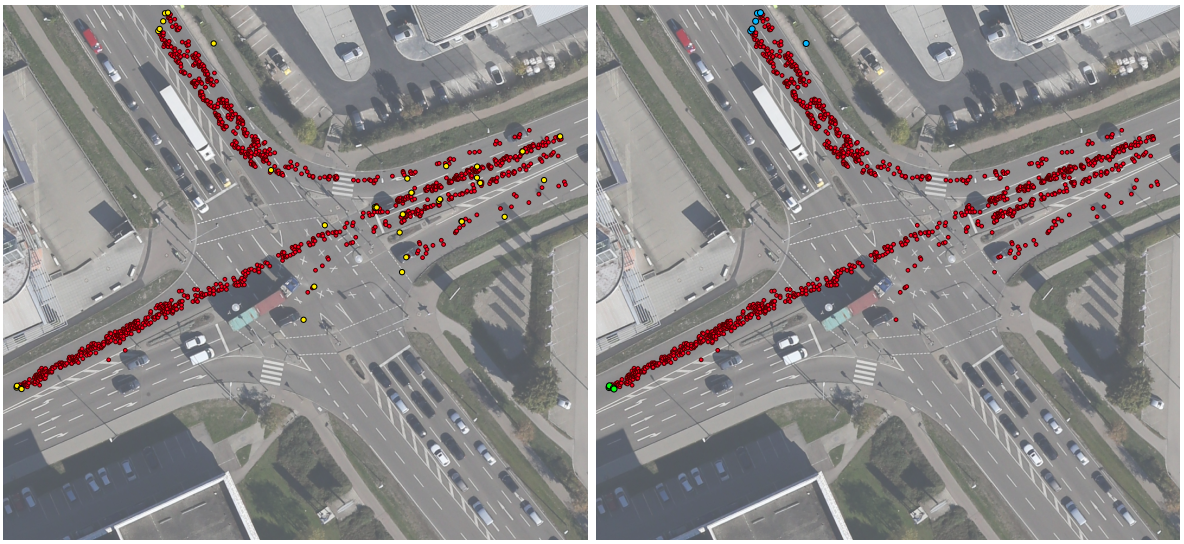
Die gefundenen Fahrspurgruppen entsprechen bereits einer guten Aufteilung der Autos - vor allem im Kreuzungszentrum können die vielen unterschiedlich ausgerichteten Fahrzeuge nun nach ihren Ursprungsrichtungen getrennt werden.

#### Finden von Endpunktgruppen

Um einzelne Fahrspuren innerhalb einer Gruppe zu finden bietet es sich an, den Spannbaum der Gruppe nochmals in kleinere Spannbäume zu unterteilen:
Alle Fahrspuren einer Gruppe besitzen zwar dieselbe Startpunktgruppe, doch die Endpunkte (welche auch von der Spannbaumsuche zurückgegeben werden) soll-

ten räumlich deutlich getrennt sein. Somit lassen sich - analog zu dem Finden der Startpunktgruppen - auch die Endpunkte einer Fahrspurgruppe nach ihrer `Navteq_ID` gruppieren.

Im linken Teil der Abbildung 4.21 ist Fahrspurgruppe 1 mit allen Endpunkten (gelb) dargestellt: Hier ist deutlich zu erkennen, dass Endpunkte auch zu Beginn oder in der



(a) Alle Endpunkte (b) Endpunktgruppen

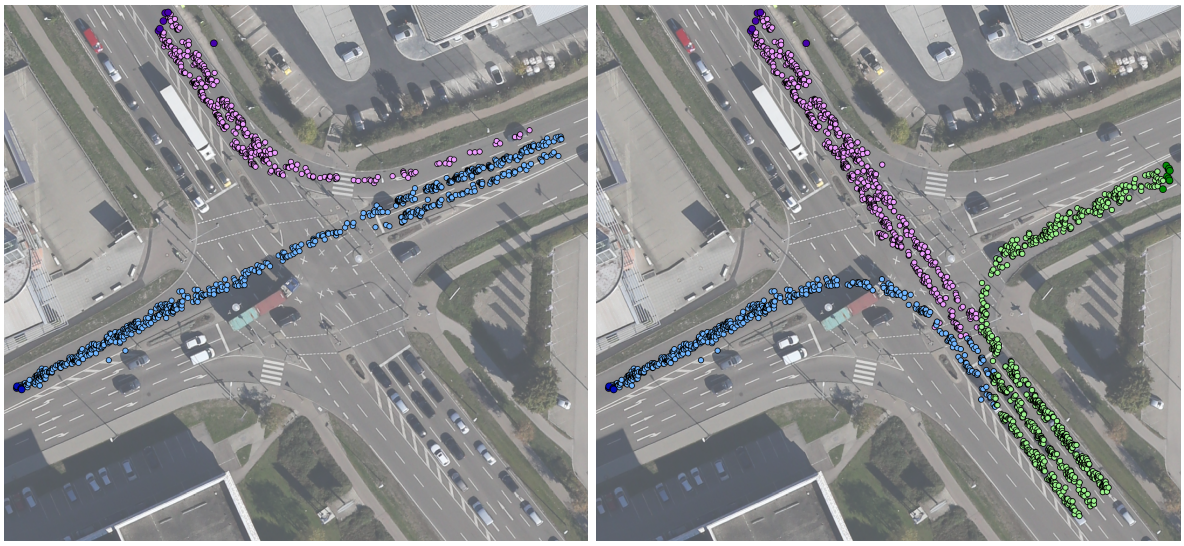
Abbildung 4.21: Endpunkte und Endpunktgruppen der Fahrspurgruppe 1

Mitte des Spannbaumes auftreten können. Da diese Endpunkte nicht hilfreich bei der Fahrspursuche sind, werden sie herausgefiltert.

Nachdem alle Endpunkte enfernt wurden, die nicht Teil der in Kapitel 4.3.1 bestimmten Randpunkte sind oder aber zu den Startpunkten ihrer Fahrspurgruppe gehören, können die Endpunkte gruppiert werden. Die Gruppen der Endpunkte von Fahrspurgruppe 1 sind im rechten Teil der Abbildung 4.21 visualisiert. Dabei sind nur zwei Endpunktgruppen (blau und grün dargestellt) entstanden.

### Umgekehrtes Durchlaufen des Spannbaumes

Die Endpunktgruppen einer Fahrspurgruppe können als Startpunktgruppen von neuen Spannbäumen verwendet werden. Mithilfe einer Breitensuche, die mit einer Endpunktgruppe beginnt und die Vorgänger-Beziehungen verwendet, lässt sich der ursprüngliche Spannbaum in umgekehrter Reihenfolge durchlaufen.
Die daraus resultierenden Spannbäume entsprechen den einzelnen Fahrspuren der Gruppe. Die beiden Ausschnitte der Abbildung 4.22 zeigen die einzelnen Fahrspuren



(a) Fahrspurgruppe 1 (b) Fahrspurgruppe 2

Abbildung 4.22: Einzelne Fahrspuren der Fahrspurgruppen 1 und 2

farblich getrennt, wobei die Endpunktgruppen in dunklerer Farbe markiert sind.
Da die `Navteq_IDs` von Straßenabschnitten mehrfach innerhalb einer Fahrspur wechseln können, kann es auch bei der Gruppierung von Randpunkten zu Problemen kommen. So können zwei formell unterschiedliche Start-/ Endpunktgruppen dennoch denselben Kreuzungsbereich beschreiben. Um diese Fälle zu berücksichtigen, werden einzelne Fahrspuren, deren Punkte zu 70% übereinstimmen, vereinigt.

### 4.3.6 Aufbau der Klasse Lane

Um die Speicherung und weitere Verabeitungen zu erleichtern, wurde eine Klasse für die einzelnen Fahrspuren angelegt. Diese ist wie in Abbildung 4.23 gezeigt aufgebaut:

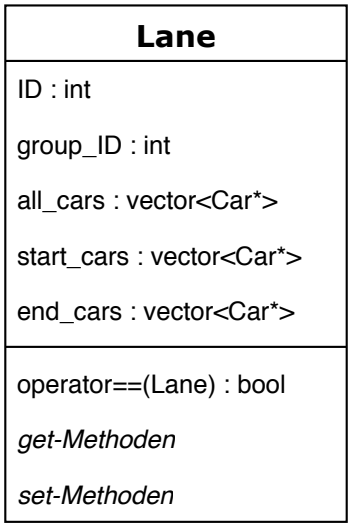



Abbildung 4.23: Klassendiagramm der Klasse Lane

Neben einer eindeutigen `ID` wird auch eine `group_ID` hinterlegt, welche die Fahrspurgruppe darstellt. Die Autos der Fahrspur werden in dem Attribut `all_cars` hinterlegt, während die Attribute `start_cars` und `end_cars` die Punkte der Startpunktgruppe und Endpunktgruppe enthalten. Der `==` Operator wurde überladen um nach der Vereinigung von Fahrspuren doppelte Elemente zu löschen.

Somit können alle Fahrspuren einheitlich erfasst und gespeichert werden. Da alle angegebenen Punkte der Fahrspur ebenfalls als `pointer` hinterlegt sind, wird unnötiger Speicheraufwand vermieden.

### 4.3.7 Trennung von mehrspurigen Bereichen

Aufgrund von Spurwechseln in den Beobachtungsdaten sowie den Effekten der Nachfolgersuche können im Bereich mehrspuriger Straßenabschnitte Autos von tatsächlich unterschiedlichen Fahrspuren zu einer einzelnen Fahrspur zählen. Dies tritt beispielsweise bei parallel verlaufenden geraden Fahrspuren oder zu Beginn bzw. Ende einer Fahrspur auf, wie auch in Abbildung 4.22 zu erkennen ist. Bevor die Punkte einer Fahrspur verwendet werden können um gemittelte Trajektorien zu berechnen, müssen die mehrspurigen Bereiche zusätzlich getrennt werden.
Ein Ansatz dafür ist die Einbeziehung der Fahrbahnmarkierungen. Diese liegen im mittleren Bereich der Kreuzung zwar nur selten vollständig vor, in den mehrspurigen Bereichen an den Rändern der Kreuzung sind diese jedoch in der Regel vorhanden. Um dies umzusetzen bietet sich eine automatische Erkennung dieser Markierungen aus den bereits vorhandenen Luftbildern an. Dass dies möglich ist zeigen Azimi, Fischer u. a. 2019. Mithilfe der Markierungen würden sich Polygone für die einzelnen Spurabschnitte erzeugen lassen, welche anschließend genutzt werden könnten, um die Punkte einer Fahrspur in diesen Bereichen aufzuteilen.

Da diese Markierungserkennung in dieser Arbeit nicht einbezogen werden konnte, wurde sie für den hier verwendeten Datensatz umgangen. Es wurde sich dafür entschieden, für jede Fahrspur ein zusammenhängendes Polygon zu erstellen, welches soweit möglich durch die sichtbaren Fahrbahnmarkierungen festgelegt wird. Somit müssen nur noch die Autos gefunden werden, welche sich innerhalb eines Polygons befinden.

Die Wahl der Polygone ist in Abbildung 4.24 anhand der Fahrspurgruppe 1 dargestellt. Dabei ist zu erwähnen, dass Abbiegespuren, die in einem zweispurigen Abschnitt en-

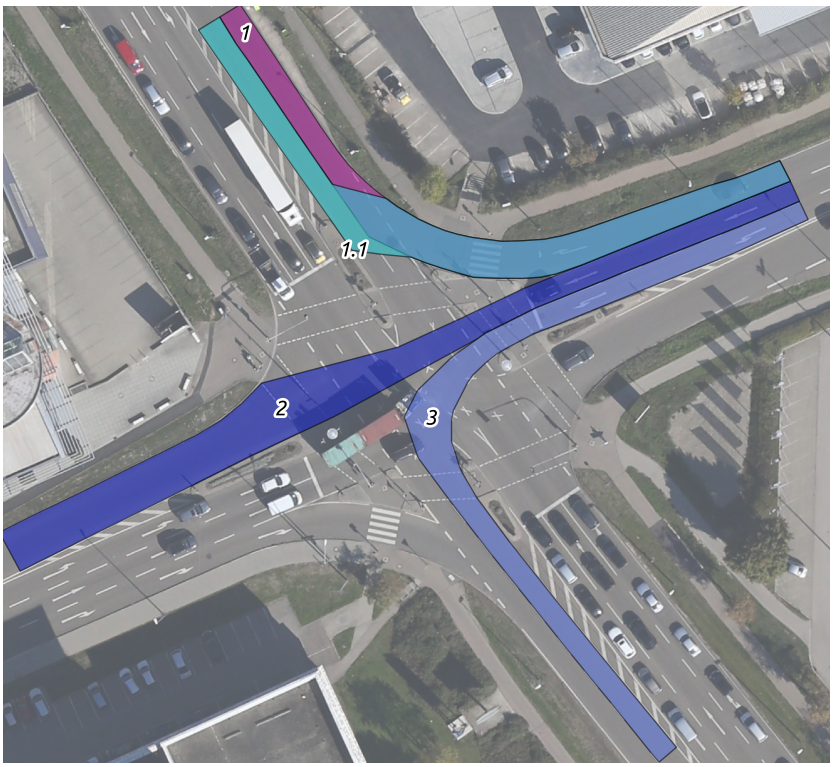



Abbildung 4.24: Polygon der Fahrspuren der Fahrspurgruppe 1

den, auf zwei verschiedene Arten durchfahren werden können. Daher wurden - wie in Abbildung 4.24 zu sehen - für diese Möglichkeiten auch zwei unterschiedliche Fahrspurpolygone (nummeriert mit 1 und 1.1) erstellt. Da diese sich zu einem großen Teil überlappen ist zu erwarten, dass die gemittelten Fahrspurtrajektorien ebenfalls sehr ähnlich verlaufen. Alle somit ermittelten Polygone sind in Abbildung 4.25 zu erkennen.

Durch die Implementierung von bereits vorhandenem C-Code eines anderen DLR-internen Projektes konnten alle Autos einer Fahrspurgruppe, welche sich innerhalb eines Polygons befinden, ermittelt und anschließend ausgegeben werden. Dafür wurden entsprechende Funktionen erstellt, welche den Import der aus QGIS exportierten Polygone, die Erstellung von benötigten Polygon `structs` sowie die Anwendung der Prüfung auf alle Autos einer Fahrspurgruppe ermöglichen. Somit lassen sich alle Au-

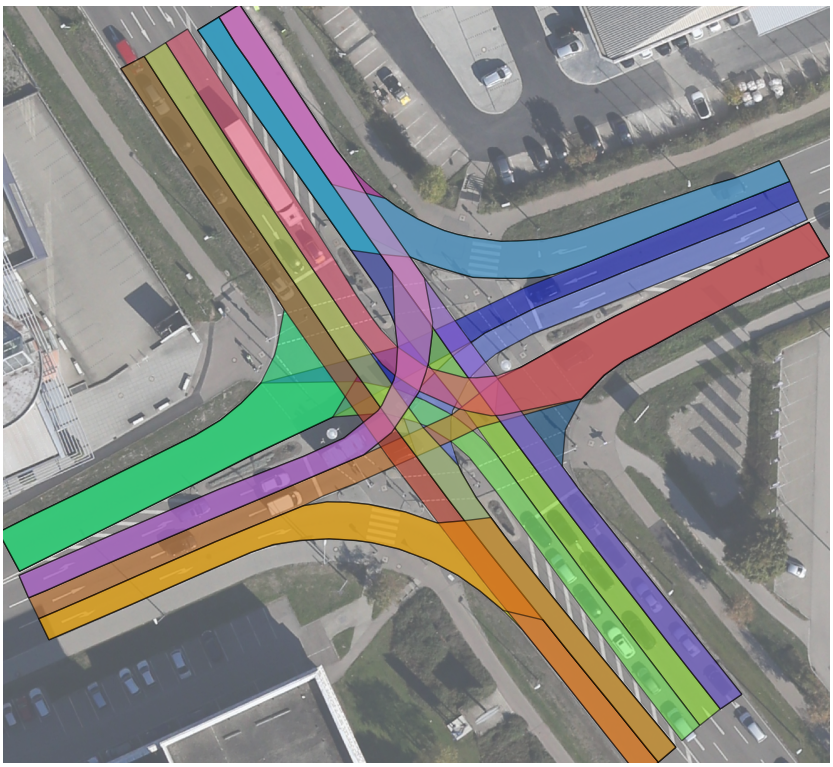


Abbildung 4.25: Polygone aller Fahrspuren

tos ermitteln, welche sich auf einer bestimmten Fahrspur befinden. Durch die vorher bestimmten Nachbarschaftsbeziehungen und automatisch erzeugten Fahrspuren ist zusätzlich sichergestellt, dass sich diese Autos auch tatsächlich entlang der gewählten Fahrspur bewegen und diese nicht einfach nur kreuzen.

Das Ergebnis dieses Algorithmus sind also Punkte, welche alle beobachteten Autos darstellen, die sich auf einer gewählten Fahrspur befinden. Zuletzt wird für jede Fahrspur eine Datei mit den Daten der zugehörigen Car-Objekte ausgegeben.

# 5 Methodik zur Erzeugung von Standardtrajektorien

Das Ziel dieser Arbeit ist es aus Luftbildsequenzen genaue Fahrspurtrajektorien für Kreuzungen zu erhalten. Die vorherigen Kapitel haben sich mit der Beschaffung der benötigten Daten, deren Bereinigung und Verarbeitung beschäftigt. Das Ergebnis sind mehrere Punktgruppen, welche die in dem Datensatz enthaltenen Autos selektiert nach den vorhandenen Fahrspuren darstellen. Um daraus für jede Fahrspur eine gemittelte Trajektorie zu gewinnen, bietet es sich an, die Punkte einer Fahrspur als Messwerte zu betrachten und durch diese eine ausgleichende Kurve zu legen.

Zunächst ist es wichtig, eine geeignete Art von Kurve zu wählen: Während nahezu gerade verlaufende Spuren gut durch eine Gerade dargestellt werden können, werden gekrümmte Spurverläufe (vor allem Abbiegespuren oder Straßenauffahrten) im Straßenbau über die Klothoide realisiert (Kasper u. a. 1968). Da das Fitten einer Klothoide an bestehende Messpunkte komplex ist wurde in dieser Arbeit eine Spline Approximation verwendet.
Auch die geradeaus verlaufenden Spuren wurden durch Splines approximiert, um eventuell leicht kurvige Bereiche innerhalb dieser Spuren berücksichtigen zu können. Somit sollten nicht nur klassisch aufgebaute Straßenkreuzungen abgedeckt sein, sondern auch komplexere Kreuzungen, die unter Umständen mehr als zwei Straßen oder kurvige Verläufe aller Fahrspuren beinhalten.

Für die Umsetzung in dieser Arbeit wurde die Software MATLAB® (Version: R2020a) von The MathWorks, Inc. verwendet (MATLAB 2020). In den folgenden Kapiteln wird

erläutert, wie eine geeignete Spline Approximation ermittelt und auf die Daten aller Fahrspuren angewendet wird.

## 5.1 Die Curve Fitting Toolbox™ in MatLab

Eine vielseitige Möglichkeit in MatLab ausgleichende Kurven zu gegebenen Daten zu erzeugen ist die Curve Fitting Toolbox™. Diese bietet neben einer App auch viele Funktionen, welche das Fitten von Kurven oder Oberflächen mithilfe von Regression, Interpolation oder Glättung ermöglicht. Für das Fitten einer Spline bietet sich die Funktion `fit()` an. Sie ermöglicht über den Parameter `fitType` die Wahl des zugrundeliegenden Models. Mögliche Modelle sind beispielsweise eine lineare polynominelle Kurve, die stückweise kubische Interpolation oder die Spline-Glättung („SmoothingSpline“). Letztere legt eine glättende Kurve durch Messdaten, wobei über den `Smoothing Parameter` $p$ und die Gewichtung $w_i$ der einzelnen Messpunkte die Stärke der Glättung und der Verlauf der Kurve gesteuert werden kann. Die mathematische Grundlage hinter der glättenden Spline ist die Minimierung des Terms:

$$p \sum_i w_i \left(y_i - s(x_i)\right)^2 + (1-p) \int \left(\frac{d^2 s}{dx^2}\right)^2 dx$$

Wobei $s(x)$ die Spline, $w_i$ die Gewichte der Punkte $(x, y)$ und $p$ den Parameter der Glättung darstellt. Die Gewichtung eines Punktes ist prozentual als Wert zwischen 0 und 1 angegeben. Der Parameter $p$ kann ebenfalls zwischen 0 und 1 gewählt werden, wobei kleinere Werte eine stärkere Glättung (bis zu einem linearen Verlauf bei $p = 0$) bedeuten. Ein Wert von $p = 1$ entspricht dagegen einer kubischen Spline Interpolation, welche durch jeden einzelnen Messpunkt verläuft. (The MathWorks 2020a, S.180-186)

Da eine geglättete Spline-Kurve sowohl gerade als auch kurvig verlaufende Fahrspuren gut approximieren kann, wurde dieses Fitting-Modell verwendet, um die gemittelten Trajektorien zu erzeugen.

## 5.2 Fitten einer Smoothing-Spline

Um die Daten aller Fahrspuren automatisiert fitten zu können, wurde ein MatLab Skript erzeugt, welches die Daten einlädt, die gewählte Spline-Glättung anwendet und das Ergebnis in eine Datei ausgibt.

### 5.2.1 Einladen und Vorbereitung der Daten

Die Punkte können am einfachsten durch eine Textdatei in MatLab importiert werden. Da die einzigen relevanten Informationen die Koordinaten der Punkte sind, werden nur diese gespeichert, wobei jeweils ein Vektor für die X- und Y-Komponente erstellt wird. Da das Fitten einer Smoothing Spline eine mathematisch eindeutige Funktion zugrunde legt, werden Trajektorien, die für jeden X-Wert mehr als eine Y-Wert annehmen, nicht korrekt approximiert. Deshalb ist es bei manchen Spuren erforderlich eine Rotation der Daten voranzustellen und somit sicherzustellen, dass das Ergebnis auch dem Verlauf der Fahrspur entspricht. Da die Kreuzung des hier verwendeten Datensatzes einfach aufgebaut ist, genügt es bei „falsch“ gedrehten Fahrspuren die X- und Y-Achse zu tauschen. Die Rotation der Daten muss vor der Ausgabe der fertigen Spline wieder rückgängig gemacht werden.

### 5.2.2 Ablauf des Fittings

Um die ggf. rotierten Messpunkte zu fitten werden in MatLab zunächst die `fitoptions` festgelegt:

```
Spline_options = fitoptions('Method','SmoothingSpline',
                            'SmoothingParam', parameter);
```

Diese sollen für die Methode `SmoothingSpline` gelten und geben den Parameter $p$ hinter `SmoothingParam` an. Das Fitting selbst erfolgt mit der Funktion `fit()`:

```
[Spline, Goodness, Output] = fit( UTM_X_rot, UTM_Y_rot,
                                  'smoothingspline',
                                  Spline_options);
```

Der Funktion werden die Vektoren der (ggf. rotierten) X- und Y-Werte, das Fitting-Modell `smoothingspline` sowie die vorher festgelegten Optionen übergeben. Die Ergebnisse sind neben der gefitteten Spline (als `cfit` Objekt) auch statistische Aussagen (`Goodness`) über die Approximation der Daten und Informationen über den Fitting Algorithmus (`Output`). (The MathWorks 2020a, S.423ff)

Da das Spline Objekt selbst nicht einfach ausgegeben und in anderer Software - bspw. QGIS - importiert werden kann, wird die Spline durch viele einzelne Punkte ausgegeben. Dabei muss der Abstand zwischen diesen Punkten entsprechend klein gewählt werden, um eine genügend genaue Rekonstruktion der Spline zu ermöglichen. Für den hier verwendeten Datensatz genügt dafür ein Punktabstand von 10cm

auf der X-Achse. Um diese Punkte zu erhalten mussten zuerst die X-Werte mithilfe der Funktion `linspace()` im Abstand von 10cm berechnet werden. Anschließend lassen sich mithilfe der Funktion `feval()` der Curve Fitting Toolbox™ die zugehörigen Y-Werte der Punkte auf der gefitteten Spline berechnen. Die somit erhaltenen Punkte müssen ggf. noch zurück rotiert werden und können anschließend in einer Textdatei ausgegeben werden.

Diese Punkte können in QGIS eingeladen werden um die gefittete Spline zu visualisieren. Dazu bietet sich ein in QGIS erstelltes Python Skript an, welches das automatisierte Einladen mehrerer Dateien ermöglichen kann. Um die gemittelten Trajektorien nicht nur als dichte Punktfolge sondern als durchgehende Linie darzustellen wurde das QGIS Plugin „Points2One“ von Pavol Kapusta (Kapusta u. a. 2015) genutzt. Dieses bietet die Möglichkeit aus Objekten eines Punktlayers automatisch Polygone oder Linien zu erstellen, welche wiederum als Shapefile gespeichert werden. Dafür sollte allerdings eine Sortierung der Punkte angegeben sein. Hier lässt sich diese einfach durch eine zusätzliche Zählerspalte in der Ausgabedatei einbinden. Somit liegen die Standardtrajektorien in QGIS als durchgehende Linien vor.

### 5.2.3 Vergleich verschiedener Fit-Parameter

Da der Verlauf einer erzeugten Spline maßgeblich durch den `Smoothing Parameter` $p$ beeinflusst wird, ist es sinnvoll, zunächst verschiedene Werte zu vergleichen. Um diesen Vergleich möglichst aussagekräftig zu gestalten ohne bereits alle Fahrspuren zu fitten, wird aus jeder Fahrspurgruppe eine Fahrspur ausgewählt, sodass alle verschiedenen Fahrspurverläufe vertreten sind. Das Ziel ist es durch diese möglichst repräsentative Auswahl den besten `Smoothing Parameter` zu finden und diesen durch

Anwendung auf die komplette Kreuzung zu validieren.

Die Polygone der gewählten Fahrspuren sind in Abbildung 5.1 dargestellt. In dieser

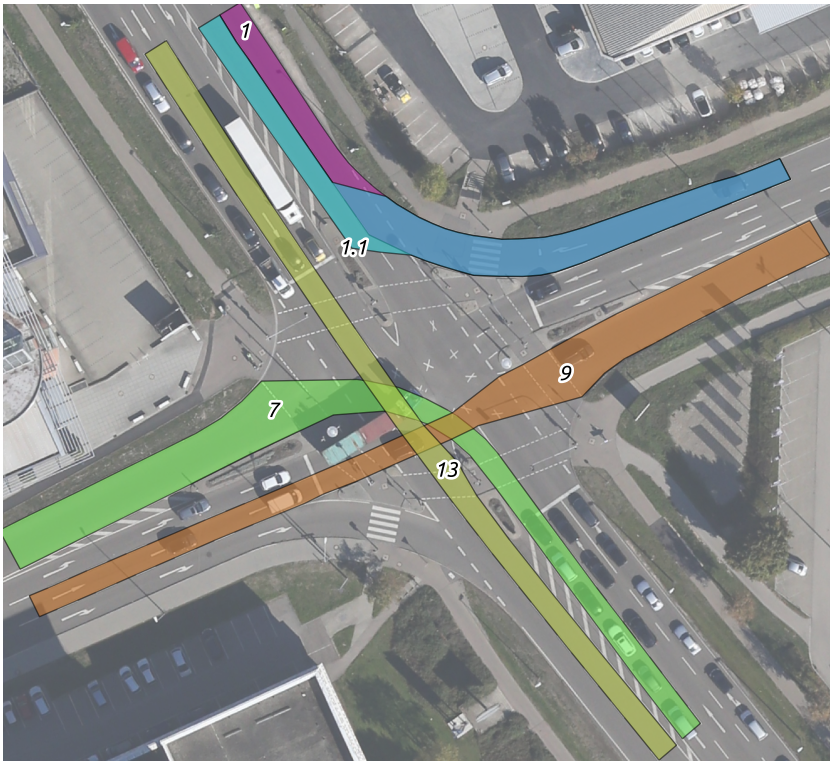



Abbildung 5.1: Polygone der gewählten Fahrspuren

Auswahl befindet sich sowohl eine Rechts- als auch eine Linksabbiegespur, wobei Erstere in einem zweispurigen Bereich endet und daher in zwei seperate Fahrspuren (1 und 1.1) aufgeteilt wurde. Außerdem wurden zwei geradeaus verlaufende Spuren gewählt. Fahrspur 13 verläuft sehr geradlinig von Nord-West nach Süd-Ost, wohingegen Fahrspur 9 im hinteren Teil einen Bereich größerer Streuung der Daten aufweist. Dieser ist vermutlich darauf zurückzuführen, dass hier zusätzlich zwei verschiedene Abbiegespuren zusammenlaufen.

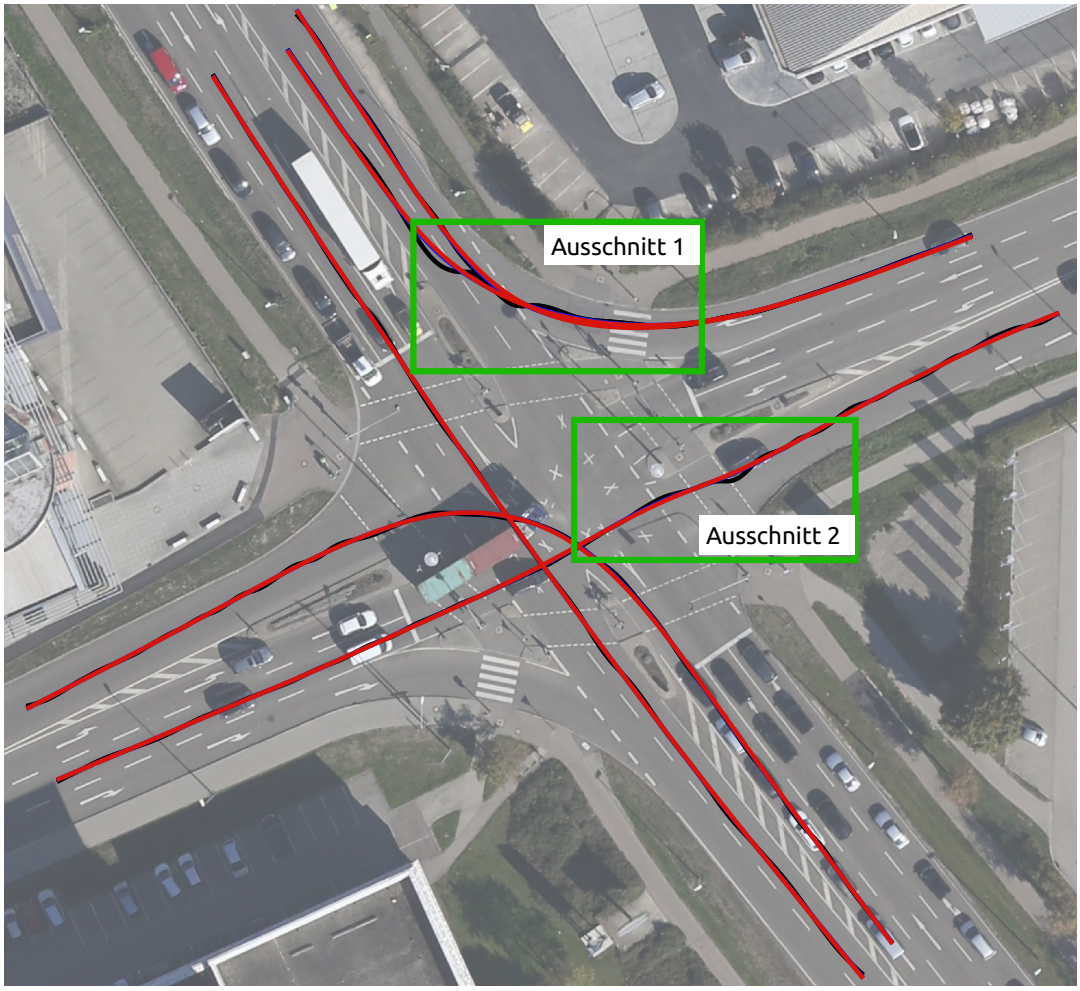
Mit dieser Auswahl an Fahrspuren wurden für jeden gewählten `Smoothing Parameter` die gefitteten Splines berechnet und in QGIS visualisiert. Dabei wurden die Messdaten sowie das Luftbild hinterlegt und beurteilt, wie gut sich der gewählte Parameter

eignet. Die folgende Auflistung zeigt die getesteten Parameter und eine kurze Bewertung der Splines:

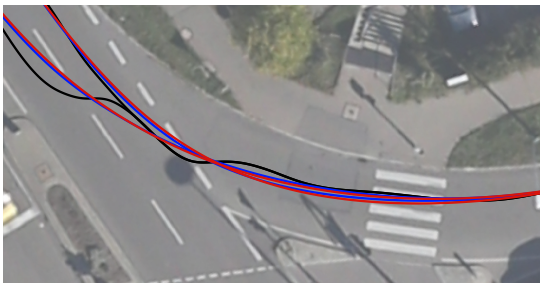
- $0.1$ : Verläuft durch zu viele Punkte und erzeugt Trajektorie mit vielen „Dellen“.
- $0.01$ : Bessere Glättung, aber zu sensibel bei gestreuten Daten. Die Fahrtverläufe wären zu unruhig.
- $0.001$ : Gute Glättung, jedoch noch etwas anfällig für die gestreuten Daten der Fahrspur 9.
- $0.0001$ : Sehr gute Glättung, jedoch verlaufen die beiden Rechtsabbiegespuren zu eng.

Da die letzten beiden Parameter bereits gut passen, wurde als Kompromiss der Parameter $0.0005$ gewählt. Dieser glättet auch die gestreuten Daten sehr gut ohne die Abbiegespuren zu stark zu glätten. In Abbildung 5.2 sind die Ergebnisse der Parameter $0.1$ (schwarz), $0.001$ (blau) und $0.0005$ (rot) dargestellt. Die Ungenauigkeiten durch den Parameter $0.1$ sind deutlich zu erkennen, das Ergebnis des Parameters $0.001$ ist dagegen an vielen Stellen kaum von dem Parameter $0.0005$ zu unterscheiden. In den beiden Ausschnitten der Abbildung 5.2 ist jedoch zu erkennen, dass der letzte Parameter ein besseres Ergebnis liefert.

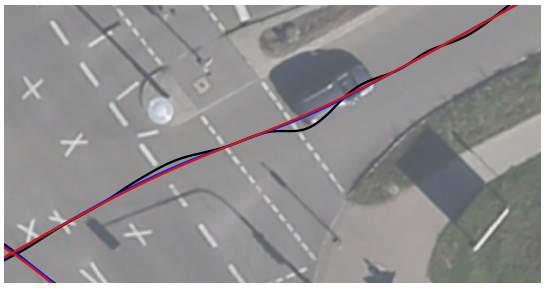
Eine weitere Möglichkeit der Bewertung könnte die berechnete `Goodness` bzw. die „Goodness of Fit“ darstellen: Diese gibt unter anderem Werte wie den an die Freiheitsgrade angepasste „R-Square“ oder den „Root Mean Squared Error“ (RMSE) an. Ersterer beschreibt mit einem Wert zwischen 0 und 1 wie gut die Daten durch die Spline angenährt werden. Der RMSE, also die Wurzel des mittleren quadratischen Fehlers, stellt die mittlere Abweichung der Daten zu der gefitteten Spline dar und




(a) Gemittelte Test-Trajektorien



(b) Ausschnitt 1



(c) Ausschnitt 2

Abbildung 5.2: Vergleich der Test-Trajektorien

sollte daher so klein wie möglich sein. (The MathWorks 2020a, S.250ff)

Beide Werte sind zwar interessante statistische Angaben, sind in diesem Anwendungsfall allerdings nicht der beste Maßstab zur Bewertung der gemittelten Trajektorien. Eine Spline, die einen großen Teil der Punkte bestmöglich annähert und somit auch einen geringen RMSE aufweist, beschreibt nicht unbedingt den besten Pfad innerhalb einer Fahrspur. Dies zeigt sich vor allem in den Werten der verschiedenen Parameter: Wie in Tabelle 5.1 zu erkennen ist, legen die Werte der beiden statis-

| | $p = 0.01$ | | $p = 0.0005$ | |
|---|---|---|---|---|
| **Fahrspur** | adj.RSquare | RMSE | adj.RSquare | RMSE |
| **1** | 0.991 | 0.936 | 0.989 | 0.995 |
| **1.1** | 0.984 | 1.140 | 0.981 | 1.215 |
| **7** | 0.997 | 0.650 | 0.995 | 0.750 |
| **9** | 0.997 | 0.792 | 0.999 | 0.925 |
| **13** | 0.999 | 0.860 | 0.995 | 0.832 |

Tabelle 5.1: Vergleich der „Goodness of Fit“ Angaben

tischen Angaben für die Spline mit dem Parameter $0.01$ ein besseres Fit-Ergebnis nahe. Der vorherige visuelle Vergleich zeigt jedoch, dass diese Spline keinen realistischen Fahrtverlauf darstellt. Daher bleibt eine visuelle Kontrolle hier die verlässlichere Bewertungsmethode.

Für die betrachtete Auswahl an Fahrspuren sind also `Smoothing Parameter` zwischen $0.001$ und $0.0001$ gut geeignet, um den Fahrspurverlauf anzunähern. Deshalb wurde der mittlere Wert $p = 0.0005$ für das Fitten aller Fahrspuren ausgewählt.

## 5.3 Ergebnis des Fittings aller Fahrspuren

Der Ablauf des Fittings aller Fahrspuren entspricht dem des vorherigen Kapitels. Alle Trajektorien werden mit denselben `fitoptions` - also einem Parameter von $p = 0.0005$ - ermittelt und ihre Punkte in Textdateien ausgegeben. Die Visualisierung erfolgt wieder über QGIS und das Plugin „Points2One“, wobei das georeferenzierte Luftbild ebenfalls hinterlegt ist. Der Überblick aller Fahrspuren in Abbildung 5.3 zeigt bereits, dass die Fahrspurverläufe gut beschrieben werden. Die Trajektorien sind in blau und rot (bei aufgeteilten Abbiegespuren) dargestellt. Um die erzeugten Trajektorien gut

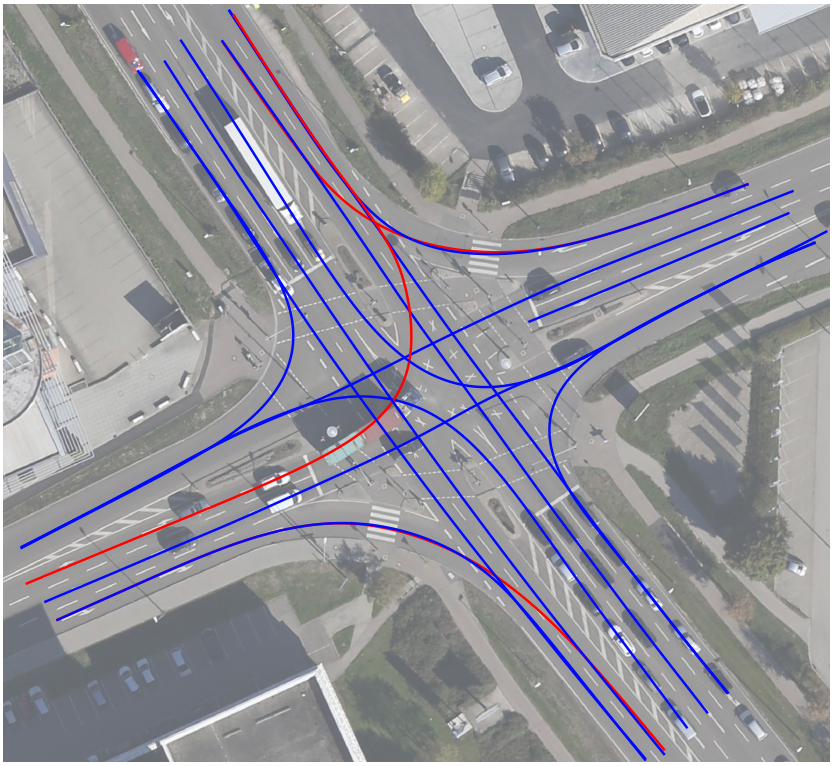


Abbildung 5.3: Standardtrajektorien aller Fahrspuren

evaluieren zu können, werden diese im Folgenden für jede Fahrspurgruppe einzeln betrachtet:

## Fahrspurgruppe 1

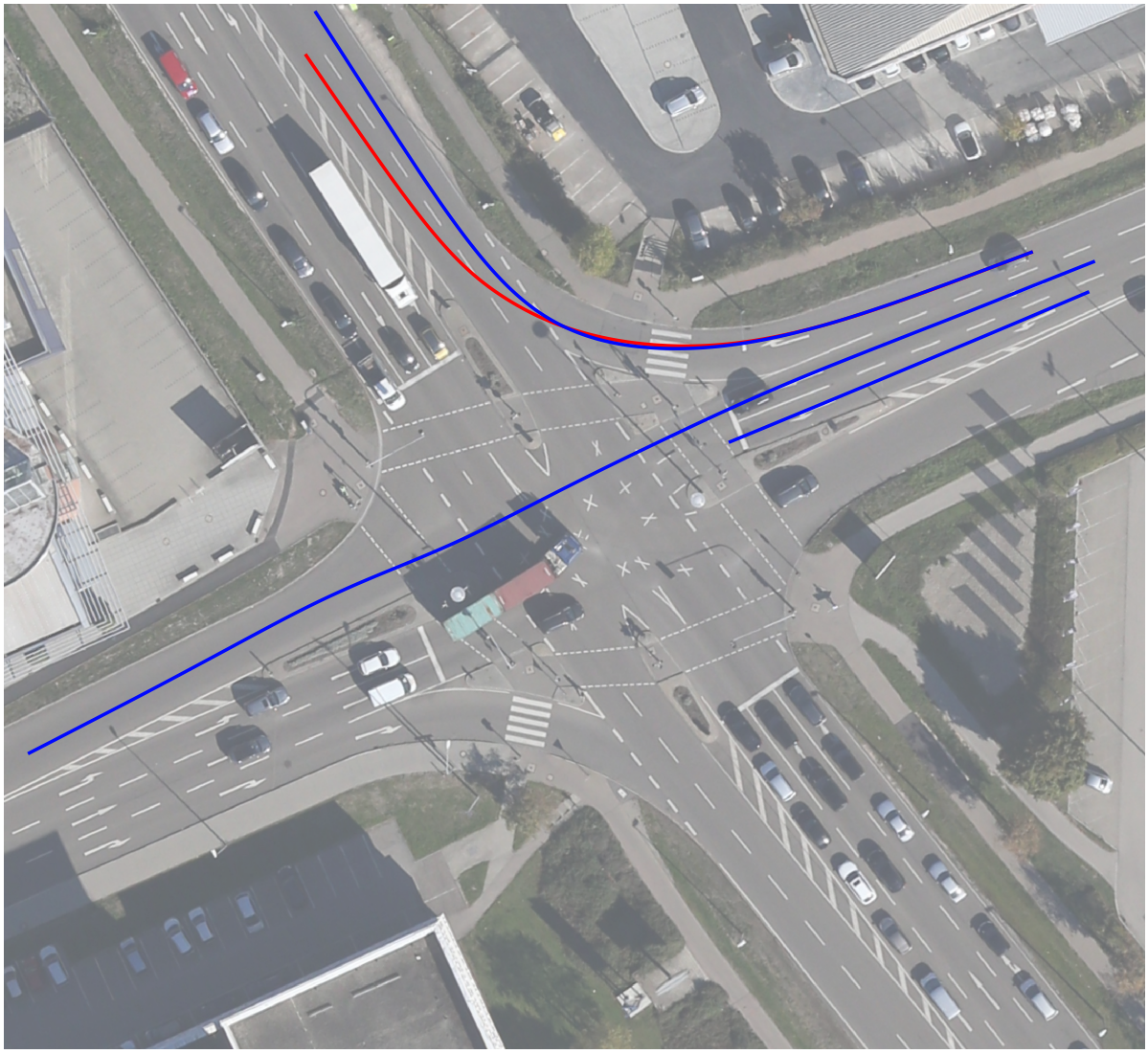


Abbildung 5.4: Standardtrajektorien der Fahrspurgruppe 1

Da die Rechtsabbiegespur bereits als Testspur verwendet wurde und die Linksabbiegespur nicht vollständig getrackt werden konnte (vgl. Kapitel 4.3.4), ist in dieser Gruppe lediglich die geradeaus führende Spur interessant. Diese verläuft realistisch, zeigt aber auch einen leicht gebogenen Bereich bei dem Zusammenlaufen mit zwei Abbiegespuren.

## Fahrspurgruppe 2

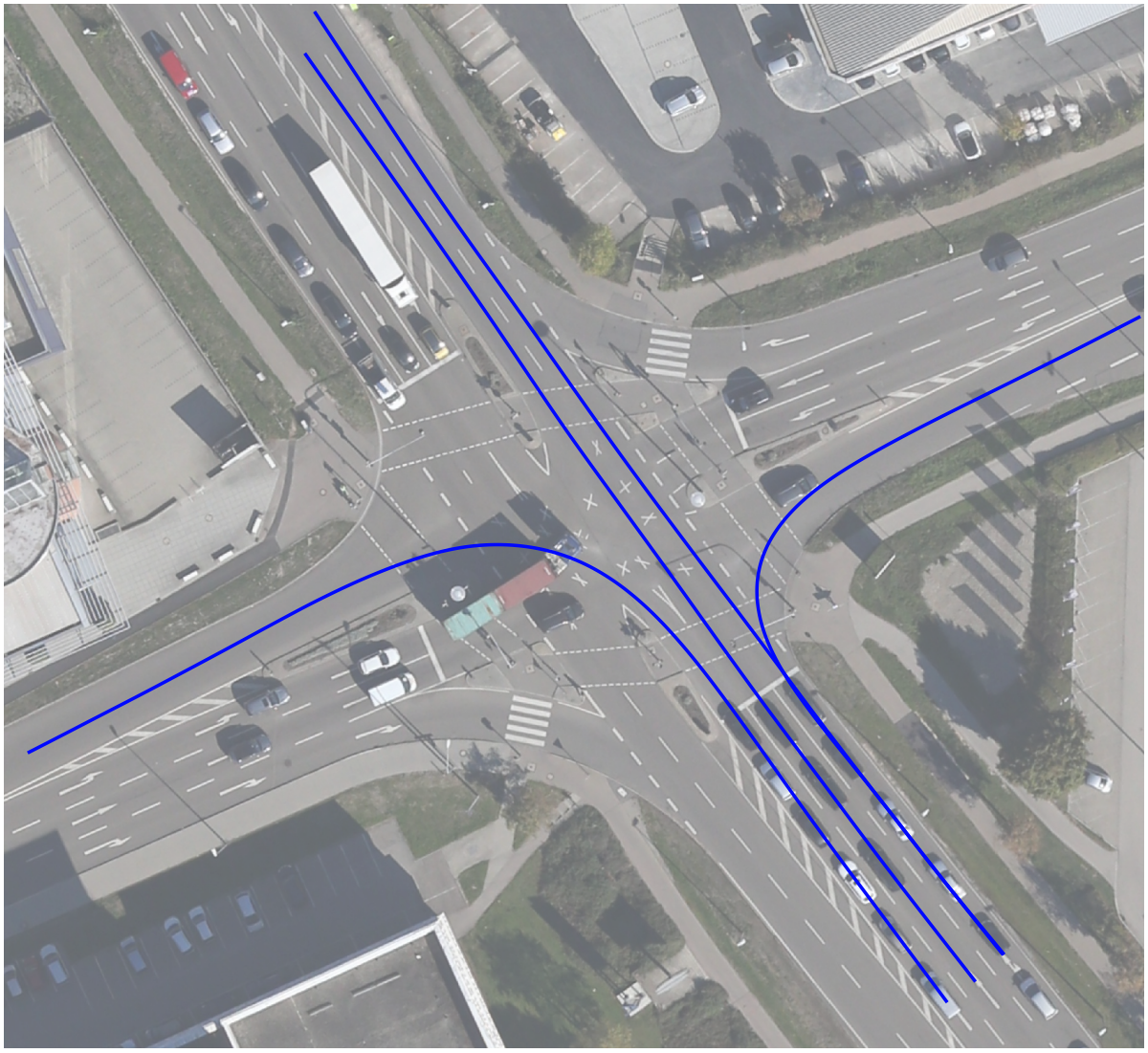


Abbildung 5.5: Standardtrajektorien der Fahrspurgruppe 2

Die Trajektorien in Gruppe 2 verlaufen allesamt sehr zufriedenstellend: Hier ist die Linksabbiegespur zwar bereits durch den Testlauf optimiert, aber auch die drei anderen Trajektorien liegen bei einem Vergleich mit dem Luftbild sehr gut.

## Fahrspurgruppe 3

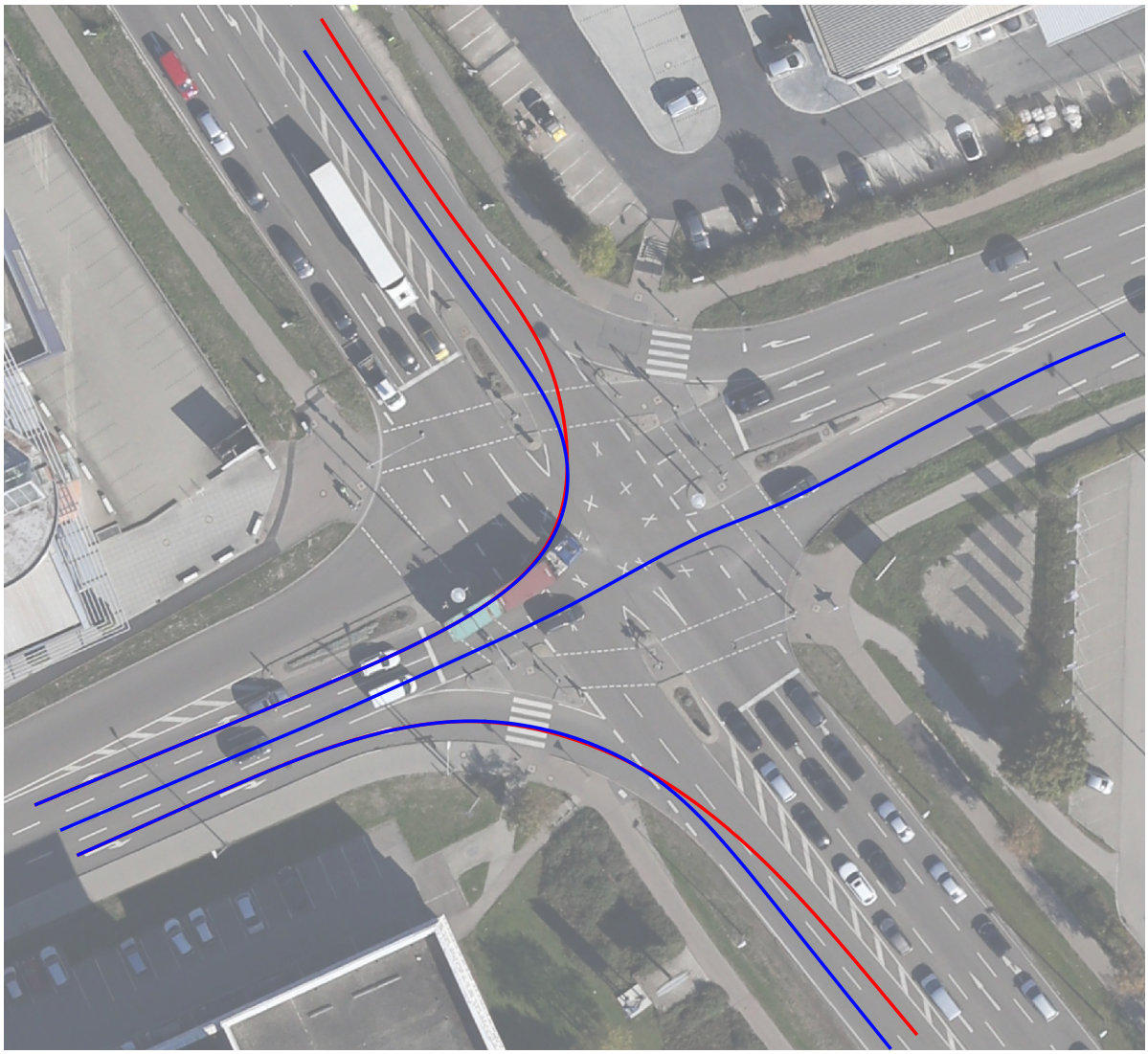


Abbildung 5.6: Standardtrajektorien der Fahrspurgruppe 3

Der gebogene Bereich der geradeaus führenden Trajektorie ist bereits in den vorangegangenen Test der Parameter aufgefallen und ist - wie auch in Fahrspurgruppe 1 - vermutlich auf das Zusammenlaufen mit zwei Abbiegespuren zurückzuführen.
Die beiden Rechtsabbiegespuren verlaufen hier auch sehr realistisch und zeigen zu Beginn der Spuren wie erwartet eine große Übereinstimmung.
Dies ist auch bei den beiden linkksabbiegenden Spuren zu beobachten. Bei diesen

fällt allerdings ein Problem der verwendeten Spline-Glättung auf: Zu Beginn des Abbiegevorgangs (im Bild auf Höhe des abbiegenden LKWs) verläuft die Spur zu eng, sodass ein Auto welches dieser Trajektorie folgt der dort vorhandene Verkehrsinsel sehr nahe kommen oder diese sogar leicht schneiden würde.

## Fahrspurgruppe 4

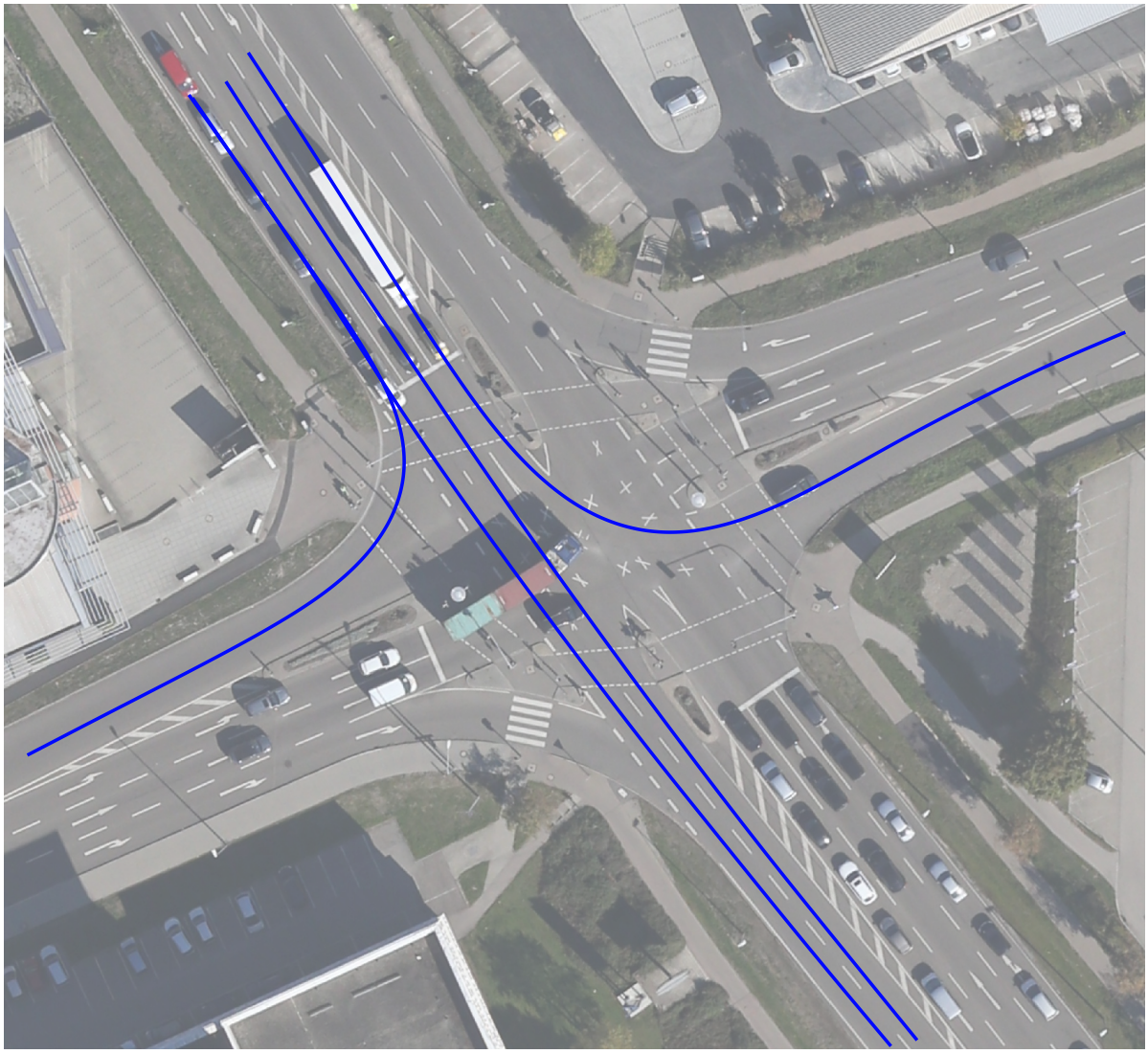


Abbildung 5.7: Standardtrajektorien der Fahrspurgruppe 4

Bei Betrachtung der Linksabbiegespur der Fahrspurgruppe 4 fällt dieses Problem ebenfalls auf: Auch hier würde ein Fahrzeug die Verkehrsinsel leicht schneiden.
Die Rechtsabbiegespur sowie beide geradeaus verlaufende Trajektorien passen jedoch wieder sehr gut mit dem Luftbild zusammen.

## Mögliche Verbesserungen

Die zu eng verlaufenden Linksabbiegespuren stellen nicht den idealen Fahrtverlauf dar. Sie sind auf den niedrigen Glättungsparameter zurückzuführen und tauchen bei größeren Werten weniger stark bis gar nicht auf.
Da ein höherer Parameter aber alle Trajektorien anfälliger für Ungenauigkeiten in den Daten macht, sollten andere Möglichkeiten untersucht werden:

**a) Gewichtung der Punkte:** Bei der Erstellung einer Smoothing Spline werden allen Punkten Gewichte zwischen 0 und 1 (Default-Wert) zugeordnet. Da die Autos innerhalb des Kreuzungsbereichs für das Ziel dieser Arbeit wichtigere Informationen liefern als die Autos am Kreuzungsrand, macht es Sinn diese auch stärker zu gewichten. Die Punkte der Ränder könnten sich außerdem mit eventuell bereits bestehenden Straßendaten überschneiden und könnten durch diese wieder ausgeglichen werden. Eine solche Gewichtung könnte beispielsweise einer Gaußverteilung folgen, indem die Punkte von den Rändern einer Fahrspur zur Mitte hin immer stärker gewichtet werden. Die Gaußverteilung in Abbildung 5.8 ist in MatLab durch die Funktion `gaussmf()` der Fuzzy Logic Toolbox™ (The MathWorks 2020b, S.505f) entstanden. Mithilfe dieser Verteilung lässt sich für jeden Messpunkt einer Fahrspur das entsprechende Gewicht berechnen.
In Testläufen mit verschiedenen Standardabweichungen zwischen 10 und 2 konnte bei der Berechung der Splines kein Unterschied bis hin zu einer Verschlechterung der Abbiegetrajektorien beobachtet werden. Damit scheint dieses Verfahren nicht für eine Verbesserung geeignet zu sein.

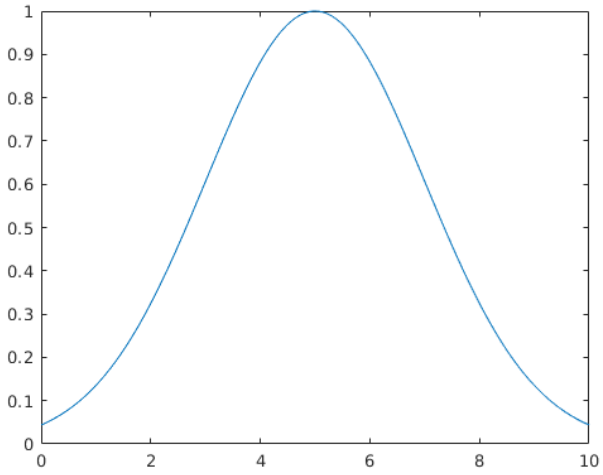



Abbildung 5.8: Gaußverteilung der Gewichtung

**b) Unterscheidung der Spurarten** Eine andere Möglichkeit wäre es, die Fahrspuren grundsätzlich nach ihrer Art oder Funktion zu unterscheiden: So könnten Abbiegespuren eine weniger starke Glättung verwenden als geradeaus verlaufende Spuren oder sogar durch eine Klothoide gefittet werden. Diese Trennung ist allerdings bei komplexeren Kreuzungen nicht immer eindeutig, da auch Fahrspuren die grundsätzlich geradeaus führen einen kurvigen Verlauf besitzen können. Eine solche Trennung könnte dennoch hilfreich sein, die Stärke der Glättung besser auf die vorhandenen Fälle anzupassen.

**Fazit der Verbesserungsmöglichkeiten:** Trotz der Probleme mit Unregelmäßigkeiten in den Messdaten oder dem zu engen Verlauf von Linkskurven lässt sich jedoch sagen, dass die erzeugten Standardtrajektorien größtenteils gut mit dem Luftbild und den tatsächlichen Pfaden der Autos übereinstimmen. Vor allem die rechts abbiegenden und geradeaus führenden Spuren, die nicht mit vielen anderen Spuren zusammenlaufen, zeigen sehr gute Ergebnisse und könnten als Grundlage für Kartendaten dienen.

## 5.4 Evaluation der Standardtrajektorien

Die geografische Genauigkeit der finalen Trajektorien wird durch die Genauigkeit der Georeferenzierung bestimmt, da über diese sowohl die Autokoordinaten als auch die Fahrbahnmarkierungen ermittelt werden. Eine fehlerhafte Position der Trajektorien kann daher nicht durch die visuelle Kontrolle mit einem hinterlegten Orthobild des Datensatzes aufgedeckt werden. Um das auszuschließen können Luftbilder oder Kartendaten aus anderen Quellen hinterlegt werden. Wie in Abbildung 5.9 zu sehen ist,

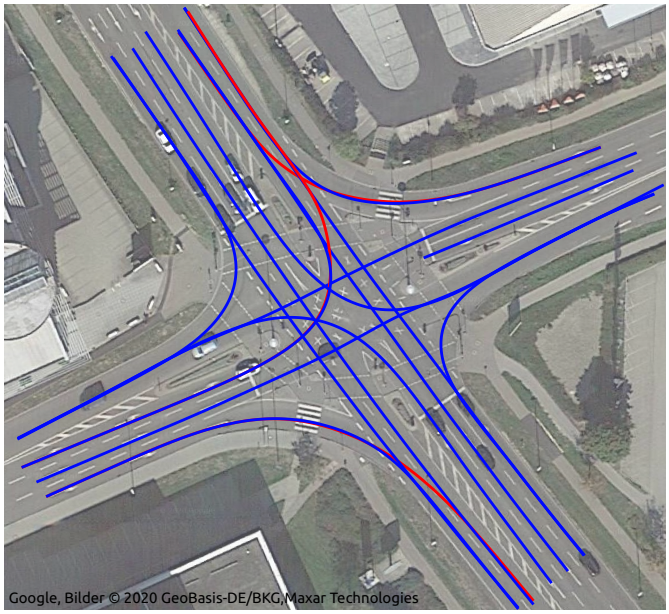



Abbildung 5.9: Vergleich der Trajektorien mit Google Satellite Bildern

zeigen die Trajektorien auch bei einem Vergleich mit den Satellitenbildern von Google einen realistischen Fahrtverlauf. Bei einem Vergleich mit den Kartendaten von OpenStreetMap in Abbildung 5.10 und Google Maps in Abbildung 5.11 fällt auf, dass beide Karten für eine Evaluation nicht genügend Details der Kreuzung kennen und die beteiligten Straßen zusätzlich nicht genau genug kartiert sind. Präzisere Möglichkeiten der Validierung wären ein Vergleich mit Daten aus HD-Maps oder sogar eine direkte Prüfung mithilfe eines hochgenau getrackten Autos.

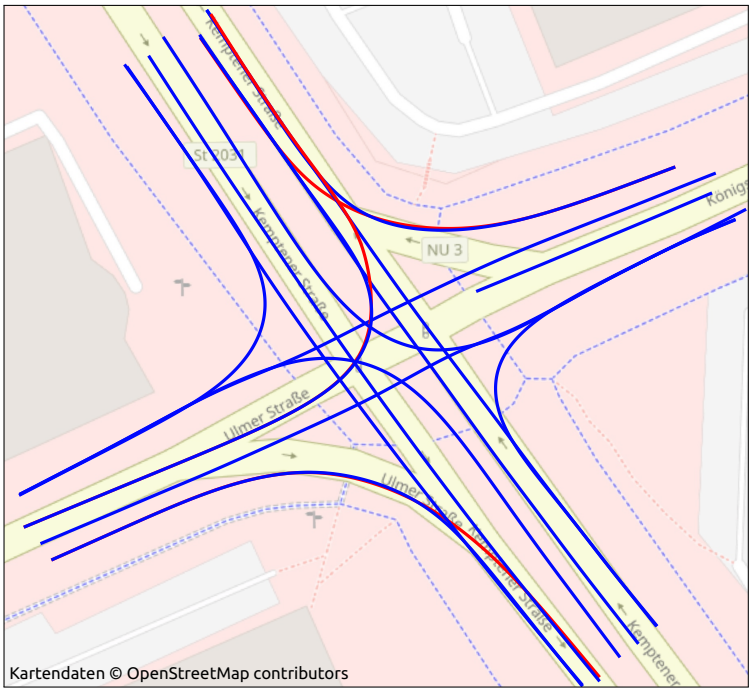



Abbildung 5.10: Vergleich der Trajektorien mit OpenStreetMap Daten

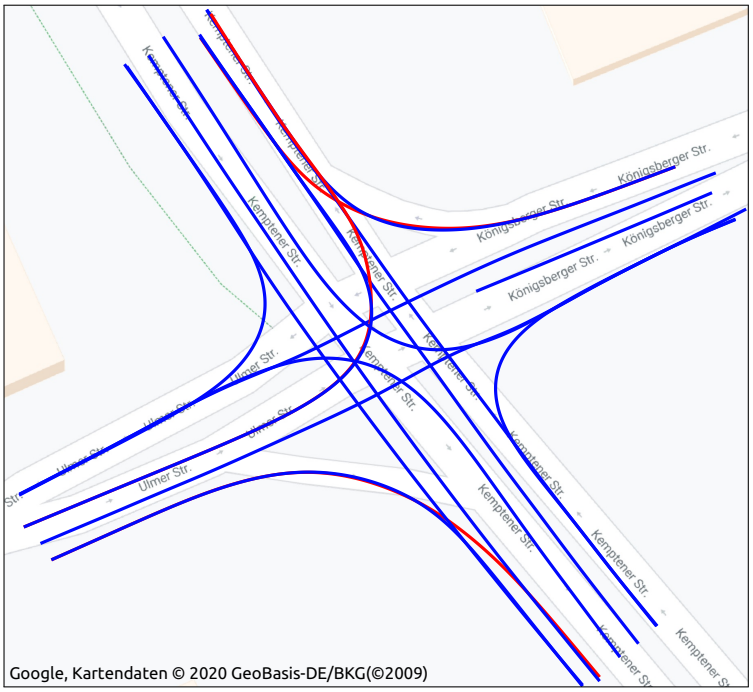



Abbildung 5.11: Vergleich der Trajektorien mit GoogleMaps Daten

# 6 Ergebnis und Fazit

In dieser Arbeit wurde zum Einen ein Algorithmus entwickelt, welcher die Daten der Verkehrserfassung des DLRs verarbeitet und die detektierten Autos nach Fahrspuren gruppieren kann. Dies gelingt im untersuchten Kreuzungsbereich trotz den Überschneidungen vieler Fahrspuren gut und ermöglicht es die Fahrspuren getrennt voneinander zu untersuchen. Desweiteren wurde eine Methodik zum Erzeugen von Standardtrajektorien entwickelt. Die Ergebnisse beschreiben den Fahrspurverlauf größtenteils präzise genug, um als Kartengrundlage für das autonome Fahren in Frage zu kommen.

Da der gesamte Ablauf zur Bestimmung von gemittelten Trajektorien auf den Daten der Verkehrserfassung aufbaut hängt der Erfolg des Algorithmus davon ab, wieviele Daten korrekt erfasst wurden. Daher ist es von Vorteil den gewünschten Kreuzungsbereich über eine lange Zeit zu beobachten und somit einen ausführlichen Datensatz zu erzeugen. Mögliche Fehldetektionen können zu einem gewissen Grad durch den Algorithmus gefiltert werden.

Ein weiterer wichtiger Aspekt ist die Wahl der Parameter für die Nachfolgersuche: Werden hier zu strenge Kriterien angesetzt, kann es zu abgebrochenen Abbiegespuren kommen. Das tritt - begünstigt durch sehr wenige Fahrzeugdetektionen - auch in diesem Datensatz auf, weshalb eine Linksabbiegespur nicht erkannt wurde. Zu großzügige Grenzen in der Nachfolgersuche können jedoch dazu führen, dass durch Fehldetektionen falsche Fahrspuren entstehen oder, dass Autos (besonders in der Kreuzungsmitte) fälschlicherweise anderen Fahrspuren zugeordnet werden.

# 7 Ausblick

Die in dieser Arbeit entwickelte Methodik bietet eine alternative Möglichkeit der präzisen Kartierung von Straßen:
Gemeinsam mit einer automatischen Erkennung von Fahrspurmarkierungen könnte eine präzise und großräumige Erfassung von Straßen aus Luftbildern möglich sein. Für Kartierungen von kleinräumigen Gebieten - wie beispielsweise einzelnen Kreuzungen - könnten sich auch vergleichsweise günstige Drohnenbefliegungen eignen. Großflächige Gebiete könnten dagegen gut durch eine Kombination von Luftbildern unterschiedlicher Befliegungen erschlossen werden, da die Fahrzeugdetektionen nicht zwingend in einem zeitlichen Zusammenhang stehen müssen. In diesem Fall würden allerdings keine Tracking Informationen zur Verfügung stehen. In Zukunft könnte es auch möglich sein, Fahrzeuge in Satellitenbildern zu detektieren, was die Kosten einer großflächigen Kartierung weiter senken könnte.

Ob die erhaltenen Trajektorien präzise genug für die Steuerung autonomer Fahrzeuge sind sollte noch in der Praxis validiert werden. Sollte dies der Fall sein, so könnten die Standardtrajektorien die bisherigen autonomen Fahrzeugeuge vor allem bei Abbiegevorgängen unterstützen bzw. diese erst ermöglichen: Das „Autosteer“ Feature des Autopilot-Systems von Tesla beispielsweise kann das Fahrzeug bisher nur innerhalb klar sichtbarer Fahrbahnmarkierungen steuern (Tesla 2020).

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Literaturverzeichnis

## Gedrukte Quellen


Azimi, S. M., P. Fischer, M. Körner und P. Reinartz (2019).
„Aerial LaneNet: Lane-Marking Semantic Segmentation in Aerial Imagery Using Wavelet-Enhanced Cost-Sensitive Symmetric Fully Convolutional Neural Networks“. In: *IEEE Transactions on Geoscience and Remote Sensing* 57.5, S. 2920–2938.

Azimi, S. M., E. Vig, R. Bahmanyar, M. Körner und P. Reinartz (2018). „Towards multi-class object detection in unconstrained remote sensing imagery“. In: *Asian Conference on Computer Vision*. Springer, S. 150–165.

Chen, W. (1997). *Graph Theory and Its Engineering Applications*.
Advanced series in electrical and computer engineering.
World Scientific, Singapur.

Jung, B. (2018). „Auf den Punkt gebracht. Von der Erdbeobachtung zum automatisierten Fahren“. In: *DLR Magazin* 159. Hrsg. von DLR, S. 6–9.

Kasper, H., H. Lorenz und W. Schürba (1968).
*Die Klotoide als Trassierungselement, 5., stark erw. Auflage*.
Dümmler, Hannover; Hamburg; Kiel; München.

Knöttner, J., D. Rosenbaum und A. Brunn (2019).
„Trennung von parkenden und am Verkehr teilnehmenden Fahrzeugen basierend auf einer automatischen Verkehrserfassung aus Luftbildern“.
In: *Publikationen der DGPF, Band 28, 2019*.

Kurz, F., T. Krauß, H. Runge, D. Rosenbaum und P. Angelo (2019).
„Precise aerial image orientation using SAR ground control points for mapping of urban landmarks“. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences-ISPRS Archives* 42, S. 61–66.

Kurz, F., D. Rosenbaum, O. Meynberg und G. Mattyus (2014).
„Real-time mapping from a helicopter with a new optical sensor system“.
In: *Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation eV* 23, S. 1–8.

Leitloff, J., D. Rosenbaum, F. Kurz, O. Meynberg und P. Reinartz (11/2014). „An Operational System for Estimating Road Traffic Information from Aerial Images“. In: *Remote Sensing* 6, S. 11315–11341.

Meyers, S. (2001). *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*. Addison-Wesley Professional Computing Series. Pearson Education, Boston.

Seifert, K. (2019). „Einarbeitung in C++ und Einlesen von Verkehrsdaten“. Projektarbeit. Hochschule für angewandte Wissenschaften Würzburg-Schweinfurt.

Turau, V. (2009). *Algorithmische Graphentheorie*. Oldenbourg Wissenschaftsverlag GmbH, München. Kap. 4.10.

## Internetquellen

*CPP-Reference: std::set* (2020). aufgerufen am: 12.02.2020. URL: https://en.cppreference.com/w/cpp/container/set.

*CPP-Reference: std::sort* (2020). aufgerufen am: 12.02.2020. URL: https://en.cppreference.com/w/cpp/algorithm/sort.

*Eigen3 Dokumentation* (2020). aufgerufen am: 21.01.2020. URL: http://eigen.tuxfamily.org/dox/.

Kapusta, P. und gezjames (2015). *QGIS Python Plugin: Points2One*. aufgerufen am: 27.04.2020. URL: https://launchpad.net/points2one.

OpenStreetMap-Wiki (2019). *DE:Tag:highway=secondary — OpenStreetMap Wiki*. aufgerufen am: 27.01.2020. URL: https://wiki.openstreetmap.org/w/index.php?title=DE:Tag:highway%3Dsecondary&oldid=1929091.

Tesla (2020). *Tesla Support - Autopilot*. aufgerufen am: 25.05.2020. URL: https://www.tesla.com/support/autopilot.

The MathWorks, I. (2020a). *Curve Fitting Toolbox™ User's Guide*. aufgerufen am 23.04.2020. URL: https://de.mathworks.com/help/pdf_doc/curvefit/curvefit.pdf.

— (2020b). *Fuzzy Logic Toolbox™ User's Guide*. aufgerufen am 05.05.2020. URL: https://de.mathworks.com/help/pdf_doc/fuzzy/fuzzy_ug.pdf.

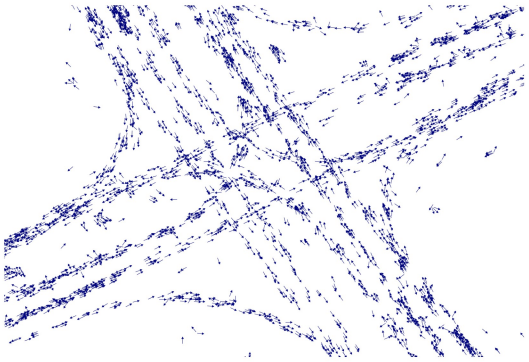
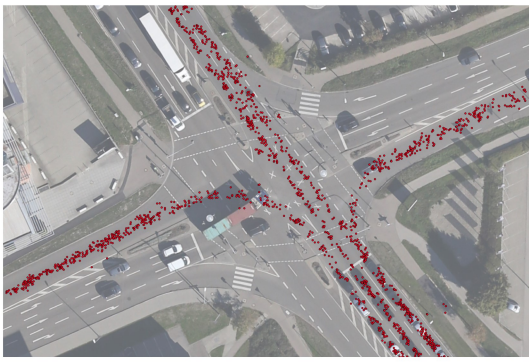
Vardhan, H. (22. 09. 2017).
*HD Maps: New age maps powering autonomous vehicles*.
aufgerufen am: 10.01.2020. URL: https://www.geospatialworld.net/article/hd-maps-autonomous-vehicles/.
Wikipedia (2020). *Orthogonalprojektion*. aufgerufen am: 21.01.2020.
URL: https://de.wikipedia.org/wiki/Orthogonalprojektion#Projektion_auf_eine_Gerade.

## Software

MATLAB (2020). *version 9.8.0.1323502 (R2020a)*.
The MathWorks Inc., Natick, Massachusetts.
Microsoft (2020). *Visual Studio Code, Version 1.45.1*.
QGIS Development Team (2020).
*QGIS Geographic Information System, Version 2.18.17*.
Open Source Geospatial Foundation.

# Bachelorarbeit im Studiengang Vermessung und Geoinformatik

FH·W·S

## Entwicklung von Methoden zur Erstellung hochpräzise georeferenzierter Karten aus Luftbildern für das autonome Fahren

**WS 2019/2020**

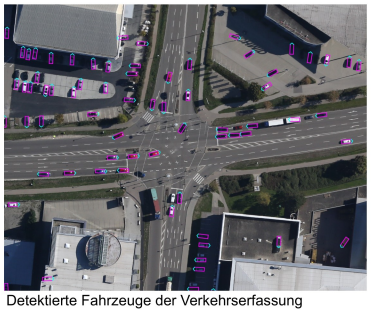
*Katja Seifert; Betreuer: Dr. rer. nat. Dominik Rosenbaum (DLR); Prof. Dr. Ing. Ansgar Brunn (FHWS)*

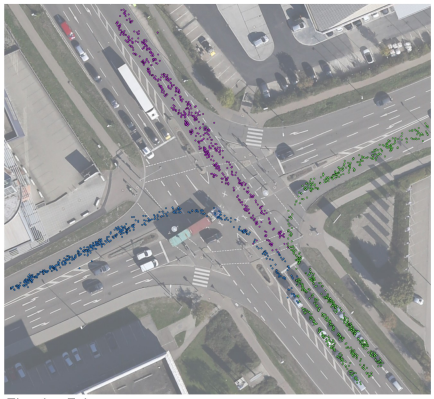
DLR

### Ziel der Arbeit:

Die Verkehrserfassung des Deutschen Zentrums für Luft– und Raumfahrt ermöglicht es in Luftbildern Fahrzeuge automatisch zu erkennen. Für diese Arbeit wurde ein Kreuzungsbereich in Senden (bei Ulm) einige Minuten mit einem Hubschrauber überflogen und alle detektierten Autos zusammengefasst.

Es sollte eine Methodik entwickelt werden, um aus diesen Daten die Fahrtverläufe aller Fahrspuren der Kreuzung in Form von Standardtrajektorien zu bestimmen.


Detektierte Fahrzeuge der Verkehrserfassung


Einzelne Fahrspuren

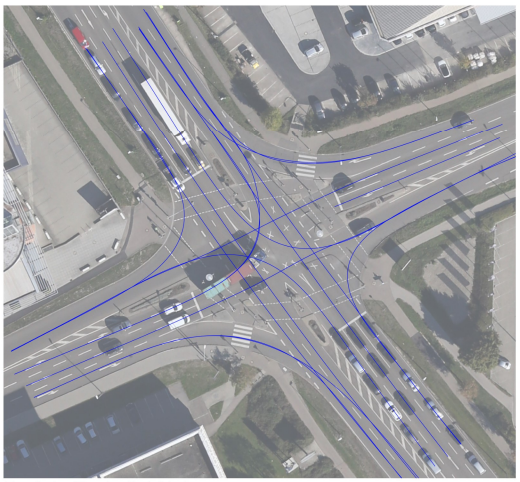
### Methodik:

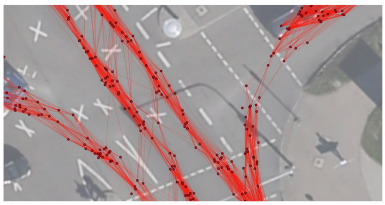
Ein in C++ entwickelter Algorithmus bereinigt die Daten und bestimmt die Fahrspuren der Kreuzung: Die Nachfolger-Beziehungen zwischen Autos werden berechnet, wodurch ein Graph aller Detektionen entsteht. Mithilfe einer Breitensuche können diejenigen Spannbäume des Graphen gefunden werden, welche die einzelnen Fahrspuren darstellen. Der Algorithmus gibt die Autos aufgeteilt auf die Fahrspuren zurück.

Die Standardtrajektorien werden durch glättenden Splines erzeugt, die mit der Software MatLab® durch die Autos gelegt werden.

### Ergebnis:

Die Standardtrajektorien zeigen realistische Fahrtverläufe und könnten als Grundlage für präzise Straßenkarten dienen. Diese können vor allem für autonome Fahrzeuge hilfreich sein, die ohne präzise Kartendaten große Kreuzungen bisher schlecht navigieren können.


Ergebnis: Standardtrajektorien


Graph der berechneten Nachfolger-Beziehungen


4K Kamerasystem an Hubschrauber BO105